

MINIMIZING ENERGY CONSUMPTION IN A WATER DISTRIBUTION SYSTEM:
A SYSTEMS MODELING APPROACH

A Thesis

by

JOHN GARRETT JOHNSTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2011

Major Subject: Civil Engineering

MINIMIZING ENERGY CONSUMPTION IN A WATER DISTRIBUTION SYSTEM:
A SYSTEMS MODELING APPROACH

A Thesis

by

JOHN GARRETT JOHNSTON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Advisory Committee,	Kelly Brumbelow
	Emily Zechman
Committee Member,	Jeff S. Haberl
Head of Department,	John Niedzwecki

May 2011

Major Subject: Civil Engineering

ABSTRACT

Minimizing Energy Consumption in a Water Distribution System: A Systems Modeling
Approach. (May 2011)

John Garrett Johnston, B.S., Texas A&M University

Co-Chairs of Advisory Committee: Dr. Kelly Brumbelow
Dr. Emily Zechman

In a water distribution system from groundwater supply, the bulk of energy consumption is expended at pump stations. These pumps pressurize the water and transport it from the aquifer to the distribution system and to elevated storage tanks. Each pump in the system has a range of possible operating conditions with varying flow rates, hydraulic head imparted, and hydraulic efficiencies.

In this research, the water distribution system of a mid-sized city in a subtropical climate is modeled and optimized in order to minimize the energy usage of its fourteen pumps. A simplified model of the pipes, pumps, and storage tanks is designed using freely-available EPANET hydraulic modeling software. Physical and operational parameters of this model are calibrated against five weeks of observed data using a genetic algorithm to predict storage tank volume given a forecasted system demand. Uncertainty analysis on the calibrated parameters is performed to assess model sensitivity. Finally, the pumping schedule for the system's fourteen pumps is optimized using a genetic algorithm in order to minimize total energy use across a 24-hour period.

This methodology results in energy savings of 5% to 13%, or \$150 to \$500 per day.

DEDICATION

to my wife, Catherine, with love

ACKNOWLEDGEMENTS

First, I would like to express my gratitude to my graduate advisor, Dr. Kelly Brumbelow. Thank you for your guidance, encouragement, and patience throughout my studies at Texas A&M. I have learned a great deal from you over the past few years, and the work presented in this thesis would not have been possible without your help.

I am also very grateful for the assistance of Dr. Emily Zechman. During my undergraduate studies, your teaching sparked my interest in water resources engineering. Your continued instruction and assistance, especially in implementing optimization methods, has been invaluable.

I would also like to thank Dr. Jeff S. Haberl. Thank you for offering your unique perspective and insight during our collaboration.

I also acknowledge and offer my sincere appreciation to my friends and co-workers at Freese & Nichols, Inc. It has been a privilege to work with you all every day and I am grateful for your ongoing encouragement.

Finally, I would like to extend heartfelt thanks to my family. To my wife, Catherine: Thank you so much for your unwavering love, support, and understanding. To my brother, Conner: Thanks for always being there for me – you have been a great encourager. To my parents, John and Kelly: I will always be grateful for your love and for the care you have continuously devoted to my academic and spiritual development. I would never have gotten this far without you.

NOMENCLATURE

dz	Difference between well WSEL and tank WSEL
GA	Genetic algorithm
gpm	Gallons per minute
HSP	High service pump
LHP	Low head pump
Q	Flow
SCADA	Supervisory control and data acquisition
VFD	Variable frequency drive
WDS	Water distribution system
WP	Well pump
WSEL	Water surface elevation

TABLE OF CONTENTS

	Page
ABSTRACT	iii
DEDICATION	v
ACKNOWLEDGEMENTS	vi
NOMENCLATURE	vii
TABLE OF CONTENTS	viii
LIST OF FIGURES	x
LIST OF TABLES	xii
1. INTRODUCTION	1
2. LITERATURE REVIEW	3
3. METHODOLOGY	15
3.1. Motivation and Objective	15
3.2. Model Formulation	16
3.3. EPANET Model Calibration	38
3.3.1. Genetic Algorithm Background	38
3.3.2. Calibration Procedure	40
3.3.3. Uncertainty Analysis	47
3.4. EPANET Model Optimization	49
3.4.1. Enumeration of Efficient Pump Combinations	50
3.4.2. Optimization Procedure	53
3.5. Methodology Summary	56
4. RESULTS AND ANALYSIS	57
4.1. Calibration Results	57
4.2. Uncertainty Analysis	64
4.3. Optimization Results	70
4.4. Caveats	74
5. CONCLUSIONS	77

	Page
REFERENCES	82
APPENDIX A	84
APPENDIX B	92
VITA	148

LIST OF FIGURES

	Page
Figure 3-1. Schematic of EPANET model	17
Figure 3-2. Raw incremental HSP station flow rates out, 09/07/08 – 09/13/08.....	18
Figure 3-3. Raw incremental flow rate into aggregate storage, 09/07/08–09/13/08	19
Figure 3-4. Aggregate storage volume from raw SCADA data, wet season.....	20
Figure 3-5. Aggregate storage volume from raw SCADA data, dry season	20
Figure 3-6. Detrended aggregate storage volume, 2006-2007 (wet season)	22
Figure 3-7. Detrended aggregate storage volume, 2008-2009 (dry season).....	22
Figure 3-8. Total system demand for Scenario 1 (2009peak2)	24
Figure 3-9. Total system demand for Scenario 2 (2009peak)	24
Figure 3-10. Total system demand for Scenario 3 (2009rise)	25
Figure 3-11. Total system demand for Scenario 4 (2008fall).....	25
Figure 3-12. Total system demand for Scenario 5 (2007flat).....	26
Figure 3-13. Tank storage volume for Scenario 1 (2009peak2).....	26
Figure 3-14. Tank storage volume for Scenario 2 (2009peak).....	27
Figure 3-15. Tank storage volume for Scenario 3 (2009rise).....	27
Figure 3-16. Tank storage volume for Scenario 4 (2008fall)	28
Figure 3-17. Tank storage volume for Scenario 5 (2007flat)	28
Figure 3-18. Characteristic and efficiency curves for pump Well3	31
Figure 3-19. Characteristic and efficiency curves for pump Well5	31
Figure 3-20. Characteristic and efficiency curves for pump LHP.....	32

	Page
Figure 3-21. Characteristic and efficiency curves for pump HSP1	32
Figure 3-22. Characteristic and efficiency curves for pump HSP2	33
Figure 3-23. Total system demand and tank storage volume for Scenario 3 (2009rise)	40
Figure 3-24. Methodology summary flowchart.....	56
Figure 4-1. Observed vs. calculated tank volumes for Scenario 1 (2009peak2)	59
Figure 4-2. Observed vs. calculated tank volumes for Scenario 2 (2009peak)	59
Figure 4-3. Observed vs. calculated tank volumes for Scenario 3 (2009rise)	60
Figure 4-4. Observed vs. calculated tank volumes for Scenario 4 (2008fall)	60
Figure 4-5. Observed vs. calculated tank volumes for Scenario 5 (2007flat)	61
Figure 4-6. Monte Carlo, Scenario 3 (2009rise), 10% uncertainty	65
Figure 4-7. Monte Carlo, Scenario 3 (2009rise), 20% uncertainty	66
Figure 4-8. Monte Carlo, Scenario 3 (2009rise), 30% uncertainty	66
Figure 4-9. Monte Carlo, Scenario 3 (2009rise), 40% uncertainty	67
Figure 4-10. Monte Carlo, Scenario 5 (2007flat), 10% uncertainty	67
Figure 4-11. Monte Carlo, Scenario 5 (2007flat), 20% uncertainty	68
Figure 4-12. Monte Carlo, Scenario 5 (2007flat), 30% uncertainty	68
Figure 4-13. Monte Carlo, Scenario 5 (2007flat), 40% uncertainty	69
Figure 4-14. Calibrated and optimized tank levels, Scenario 3 (2009rise)	71
Figure 4-15. Calibrated and optimized tank levels, Scenario 5 (2007flat).....	72

LIST OF TABLES

	Page
Table 3-1. Pump summary table	34
Table 3-2. Example set of 33 decision variables for model calibration.....	43
Table 3-3. Example set of 17 decision variables for model calibration.....	46
Table 3-4. Example set of 4 decision variables for model optimization.....	54
Table 4-1. Calibrated set of EPANET model parameters.	58
Table 4-2. Optimized parameters and energy statistics, Scenario 3 (2009rise)	71
Table 4-3. Calibrated parameters and energy statistics, Scenario 3 (2009rise)	71
Table 4-4. Optimized parameters and energy statistics, Scenario 5 (2007flat).....	72
Table 4-5. Calibrated parameters and energy statistics, Scenario 5 (2007flat).....	72

1. INTRODUCTION

For the average U.S. public water utility, total operating costs are dominated by the day-to-day cost of operating pumps. In general, this economic cost can be reduced by operating each individual pump at or near its best efficiency point, by making good use of available elevated storage to smooth out peaks in water demand, by keeping operation time of each individual pump to a minimum, or by prioritizing pump operations at night if the energy provider charges according to a diurnal pricing scheme. In addition to the simple economic cost of running pumps continuously, one might also consider the environmental cost caused by excessive emissions of greenhouse gases. This environmental cost is also minimized by lowering total energy use, although for the purposes of calculating environmental cost, diurnal energy pricing patterns would be irrelevant. Both goals of cost minimization can conflict with the goals of providing adequate water pressure to customers and of keeping a certain minimum volume of elevated storage reserved in case of fire emergency.

The goal of this thesis is to develop and calibrate a computer model of a real-world water distribution system (WDS) with a view toward minimizing energy use or energy cost on a daily-to-weekly basis. The WDS model parameters are calibrated based on historical pumping and storage data.

This thesis follows the style of *Journal of Water Resources Planning and Management*.

Given the forecasted flows for an upcoming period, the optimization function finds the best set of pumped flow rates over discrete time intervals. These flow rates, when combined with a predetermined table of efficient pump operation schedules, will yield the pump operating parameters for simulation. A longer-term energy use optimization study could also incorporate the cost of future infrastructure upgrades and long-term pump maintenance costs, but this work is focused on optimizing the weekly pump operating schedule itself.

The calibration and optimization routines presented here were developed using a genetic algorithm (GA) as the basis for repeatedly evaluating model output and finding improved model parameters. Other optimization algorithms could be adapted for use with these routines, and some may be more effective than a GA at exploring the decision space to find optimal solutions.

2. LITERATURE REVIEW

Current literature on cost minimization for WDS pumping offers a variety of approaches to minimizing operational costs without sacrificing performance. The specific criteria for measuring performance of a WDS will vary depending on the system being studied, but in general a well-performing system provides a certain minimum water pressure to all customers throughout the day, maintains a reasonable volume of storage in elevated tanks, and is able to satisfy water demand during emergencies and peak usage times. A minimum measure of water quality can also be quantified by water age or by parts-per-million of chlorine. Typically, an optimization algorithm uses an external hydraulic simulation model, such as the freely-available EPANET (Rossman 2000), to calculate these hydraulic properties. Other WDS performance criteria can include long-term effects of pump scheduling on, for example, maintenance or capital improvement costs; however, these kinds of costs are more difficult for the modeler to assess or quantify.

A modeler attempting to optimize WDS pumping schedules typically uses an algorithm to generate many solutions comprising a variety of pump schedules, that is, the times that each pump should be operating and/or the speed at which it should operate. For each solution, this algorithm will measure the fitness of each solution according to an objective function defined by the modeler. For each solution, this objective function may return the energy required in kilowatt-hours, the operational cost in dollars, or some combination. The general goal is for this fitness value to be minimized by randomly or incrementally adjusting the numbers describing pump

combinations, keeping the good solutions and discarding the bad ones, until an apparent minimum cost is discovered. However, the mathematically complex nature of any WDS means that an exhaustive technique of finding a globally optimal solution is infeasible. Even with today's computing power, a fully exhaustive simulation and collation of the results of every possible pump combination with every possible customer usage pattern for a particular WDS would likely require years of continuous computation. Instead, a modeler will implement an algorithm to explore general areas within that "decision space" of various pump combinations until a locally optimal solution is found. Repeated execution of the algorithm may yield a set of completely different locally optimal solutions that the modeler can compare. Such an algorithm searching for an ideal solution is often compared to a mountain climber searching for the latitude and longitude of the highest point in a mountain range, that is, changing two variables within certain limits (the decision space) to maximize elevation (the objective function). The decision space for any optimization has a number of dimensions equal to the number of variables being tweaked in search of a good solution, so a pumping optimization problem considering several pumps across several days' time will have a much more vast decision space than a simple map of a mountain range.

A variety of generalized optimization algorithms are available in the literature, progressing from non-linear programming to genetic algorithms and ant-colony optimization. Depending on the way the problem is formulated, any one or even any combination of methods like these can be used to find a locally-optimal solution with minimal operational costs. Computation time can be further reduced if the complicated

behavior of the WDS or the optimization algorithm is somehow simplified. The literature referenced here is intended to chart the development of various notable approaches to operational cost minimization, highlighting a variety of ways to formulate operational costs, WDS performance, optimization techniques, and simplifying assumptions.

Brion and Mays (1991) developed a methodology for minimizing pumping costs using a general purpose non-linear programming optimization algorithm. At that time most approaches to pumping optimization had been formulated using dynamic programming (DP), which scales poorly as the number of decision variables rise; that is, DP approaches only worked well for simple WDSs with few hydraulic elements. Brion and Mays instead used the non-linear programming optimization algorithm GRG2 to minimize total pump operation costs during a single day, linking it with hydraulic simulation program KYPipe to enforce mass/energy balance hydraulic constraints within the WDS. The decision variables specified for the GRG2 algorithm were a series of decimal values assigned to each pump, each describing the length of time that pump was to operate during the hydraulic time step (Brion and Mays 1991). This methodology required extensive simplifications to be applied to the WDS model and its solutions tended to prescribe excessive on/off pump cycling, which is infeasible as that would drastically reduce the life of the pumps. An experienced pump operator, however, could manually adjust such infeasible solutions to avoid pump cycling.

Ormsbee and Lansey (1994) reviewed several contemporary approaches to the optimal control of pumps in a WDS and categorized three important components of an

optimization approach: a hydraulic network model, a demand forecast model, and an optimal control model. The authors emphasized the potential of improving the network model and demand forecast model by integrating them with a supervisory control and data acquisition system (SCADA), which water utilities use to monitor certain hydraulic properties of network components in real time. The authors also mention that the operating cost of WDS pumps is variable, as most energy providers charge a flat fee based on kilowatt-hours consumed and a surplus demand fee during peak energy usage time intervals. Thus, if an optimization routine's objective function is equal to total pump operating costs, and not simply the total amount of energy used, the modeler must make a special modification to the objective function that accounts for the varying unit cost of energy. This further restricts an already complex decision space, increasing the difficulty of optimization. The expected cost of equipment maintenance and replacement is another facet of optimization that is difficult to explicitly integrate into the objective function, and is generally controlled implicitly, e.g. by choosing large hydraulic time steps for the simulation or by limiting the total number of pump switches across the simulation (Ormsbee and Lansey 1994).

Water quality can also be considered when optimizing pumping schedules. Sakarya and Mays (2000) introduced a constraint on water quality and compared the results of three different objective function approaches in a non-linear pumping optimization algorithm. The first minimized the deviation of a given chemical concentration from the goal concentration, the second minimized total pump operation time, and the third minimized total energy costs. A simple hypothetical WDS was used

to test the optimization, and as with most modern WDS optimization schemes, the hydraulic constraints were explicitly satisfied through the use of EPANET as a simulation model. The authors mention that other studies commonly enforce the periodicity of tank levels and node pressure by applying tight restrictions to tank storage levels at the end of a 24-hour period, but that this strategy is ineffective in ensuring periodicity of water quality. Instead, the authors chose a sufficiently long simulation time period such that long-term hydraulic and water quality steady state conditions could be observed for any given pump schedule. This increased computation time, but implicitly satisfied periodicity requirements for both volume and quality (Sakarya and Mays 2000).

Genetic algorithms were first introduced to the least-cost design of a WDS by Simpson et al. (1994). The authors did not consider the optimization of operating costs specifically, but some general principles of least-cost optimization can be gleaned from the paper nonetheless. The GA developed by the authors generated solutions described by a binary string. Each 1 or 0 in the string denoted a yes-or-no decision on a particular design option, like “pipe [1] has to be cleaned” or “pipe [4] has a parallel pipe of diameter 356 mm” (Simpson et al. 1994). A predetermined cost was associated with each design option, and additional penalty costs were assessed for minimum pressure constraint violations. The authors selected a GA to find optimal solutions because of its ability to generate a diverse population of alternative solutions in a large decision space. The results of the GA were compared to a non-linear technique which generated one solution at a time, beginning from a single starting point. The GA consistently provided

locally optimal solutions, and was even able to find the global solution if allowed to run for long enough. Its solutions were found to be comparable to solutions provided by a general-purpose non-linear optimizer, though it took longer to reach those solutions and required the decision variables to be simplified from continuous diameter variables to binary form (Simpson et al. 1994). Because decision variables in a pumping optimization problem are typically continuous, for instance the number of hours a pump should operate or the speed at which it should operate, this approach to GA formulation would require similar binary simplifications in order to succeed. The non-linear optimizer used by Simpson et al. was not encumbered by the same limitations. However, the GA was also able to generate a whole population of viable alternative solutions, while the non-linear optimizer could only generate one at a time; the GA's ability to generate a diverse class of solutions offsets the shortcoming of its longer computation times.

More recently, direct search methods, or so-called "hillclimber strategies," have been used by van Zyl et al. (2004) to refine pumping schedules provided by a GA. Although genetic algorithms are good for quickly finding a near-optimal solution, they do not generally perform well in zeroing in on a local optimum. To address this shortcoming, the authors employed two direct search methods – the Fibonacci coordinate method and the Hooke & Jeeves pattern search method – in order to refine the final solutions provided by the GA. A GA's stochastic approach excels at finding reasonably optimal solutions quickly. But by stopping the GA when a feasible solution is found, and then systematically changing each decision variable in small steps,

repeatedly evaluating the objective function, and moving along the steepest gradient toward more optimal solutions, a direct search algorithm can find the local optimum solution in the small decision space surrounding a GA's solution. A hybrid combination of GA and direct search methods was found to converge to an optimum solution much more rapidly than a GA by itself. In a case study of a particular WDS, the hybrid method was able to converge to an optimum solution in only 8,000 objective function evaluations, while the GA alone took 200,000 evaluations to converge to a comparable solution (van Zyl et al. 2004).

Another tool that can be implemented to save computation time is dynamic programming optimization. Ulanicki et al. (2007) formulated a two-step WDS pumping optimization approach that first used a generic non-linear programming algorithm to minimize operating costs of a WDS with relaxed constraints. Using solutions from this first stage, the authors were able to use a dynamic programming algorithm to very quickly develop new solutions with different starting conditions or other constraints (Ulanicki et al. 2007).

Long-term tradeoffs between WDS design cost, operating cost, and reliability when evaluating the fitness of a particular WDS design were considered by Farmani et al. (2005). The paper introduced a complex multi-objective approach to optimizing for two variables simultaneously: a low capital construction cost and a small "maximum individual head deficiency," that is, the risk of not meeting demand for water. These two goals are in direct opposition to one another. Instead of taking one of the traditional approaches to optimization, e.g. minimizing cost while penalizing for high head

deficiencies or minimizing head deficiency while penalizing for high costs, the algorithm generates a spectrum of optimal solutions along a Pareto front, that is, a set of optimal solutions for the two objective functions. At one end of the Pareto front, costs are low while head deficiency is high, and at the other end, vice versa. By avoiding the explicit assignment of economic value to head deficiency and instead presenting a smooth tradeoff curve between cost and deficiency, this method enables a decision-maker or WDS operator to use their own judgment when choosing a particular design or operating schedule (Farmani et al. 2005). While this paper focused on the design of new WDSs or the expansion of existing WDSs, the approach of simultaneously optimizing two conflicting objectives could also be adapted to pump scheduling problems, for example to generate a Pareto front of minimized operational costs vs. maximized average tank levels. This Pareto approach could provide a WDS operator with a more detailed understanding of the tradeoff between cost and reliability.

Ant-colony optimization has recently been implemented by Lopez-Ibanez et al. (2008) to minimize electrical cost and, implicitly, pump maintenance costs in a WDS. Rather than represent a pump schedule using binary variables, the authors represented the schedule by defining a series of integers corresponding to the hours each pump will spend in either an on or off state. This schedule was implicitly limited by a “time controlled trigger” integer representing the maximum number of switches between on and off for each individual pump. So, for a maximum number of switches equal to five, one pump’s schedule across the simulation period would involve three on cycles and three off cycles. By limiting the number of pump switches in this way, one can

implicitly avoid unrealistic pump schedules involving rapid pump cycling, which may shorten the lifespan of pump equipment. Other WDS optimization schemes which formulate pump scheduling decision variables as on/off binary values at large, discrete time steps are unable to generate short-time-step solutions without tending to cycle pumps rapidly; however, the authors' approach successfully limits pump switching without sacrificing the ability to describe the pump schedule in 1-hr time increments. Another distinguishing feature of this work is the omission of penalties applied to the objective function when constraints are violated. Because EPANET is used as the simulation framework, hydraulic constraints are always satisfied, but no other constraint violations result in penalties. Instead, optimal solutions are found by keeping and ranking all solutions according to criteria of descending importance: first, node pressure requirements, then simulation warnings, storage tank volume deficits, and finally low objective function values, i.e. low energy usage. This sorting approach allows the operator to avoid having to set and tweak arbitrary penalty values for each type of violation. According to the authors, this non-penalizing ant-colony approach generates better solutions than a stock genetic algorithm, and does so more quickly (Lopez-Ibanez et al. 2008).

Finally, rationing of water use in emergency situations was considered by Jeong and Abraham (2009). When the ability of a WDS to satisfy full water demand is compromised, whether by intentional attack, main breaks, or extended drought conditions, a rationing plan can be developed to ensure continued water service to "critical facilities" such as hospitals, emergency response and communication centers,

and energy providers. Any minimization routine for use in a time of crisis should be able to develop its solutions quickly. Jeong and Abraham chose to implement a GA alongside the EPANET hydraulic solver for minimization of several “consequence indices,” including the degree of water supply disruption to the critical facilities, economic loss experienced by reduced output of industry, and the number of residents affected by the water outage. Each of these consequence indices was expressed as a function of the percent of total water demand satisfied; for example, a hospital receiving 80% of its water demand could be assigned a higher consequence index than a large group of residents receiving only 50% of their water demand. The decision variable of this optimization process was a set of rationing multipliers, one for every 4-hour period, applied to the 24-hour average water demand for each customer. A discrete set of water rationing plans was devised before executing the optimization, supplying water to the customer either 24, 8, or 4 hours a day. The initial population of the genetic algorithm was randomly drawn from a database containing these plans, which were then evaluated for hydraulic feasibility using an EPANET model of the damaged WDS. The fitness of a particular population set was a combination of the consequence indices for all customers and a penalty function applied for non-hydraulically-feasible solutions. Solutions were then plotted on a three-dimensional Pareto graph showing the impact on critical, industrial, and residential customers. The Pareto graph output allows decision-makers to evaluate the tradeoffs for any given disaster scenario (Jeong and Abraham 2009). One limitation of this work included a lack of hydraulic controls or fitness function penalties regarding water levels in elevated storage tanks, meaning that an

“optimal” solution in the test case would drain all tanks completely in three days. Any rationing plan developed using this methodology would only be suitable for use during that period, as a plan to be used beyond three days would require maintenance of some minimum storage volume.

In summary, the current literature on energy and cost minimization for WDSs offers a variety of model formulation techniques and optimization methods. Early on, hydraulic models were generally formulated as a set of simplified linear or non-linear equations, but with the advent of powerful desktop computing, this method of formulation has been supplanted by full hydraulic network simulation through an external program, such as the U.S. Environmental Protection Agency’s EPANET (Rossman 2000). Certain independent variables describing physical and operational parameters of the hydraulic network are formulated and fed to some type of fitness function. This function measures some aspect of the WDS’s performance – ability to meet system demand, ability to supply adequate water pressure, operational cost, energy used, etc. Some method of optimization is then used to minimize or maximize the value of this fitness function by selecting an ideal set of independent variables subject to certain restrictions. The exact formulation of these variables and the methods for selecting their optimal values are up to the modeler; however, the literature shows that there are a variety of effective ways to generalize the network model and its input variables to speed computation time and increase the likelihood of finding an optimum. For instance, simplification can be accomplished by reducing the hydraulic complexity of the system (that is, the number of pumps, pipes, and storage tanks), by lengthening the

simulation's hydraulic time step, by directing the optimization process toward a systematic, logical search of the decision space and away from time-consuming random guesses, or by linearizing complex equations in the optimization process.

3. METHODOLOGY

3.1. Motivation and Objective

This work aims to develop a general methodology that can be easily implemented by WDS operators to (a) create and calibrate a simple EPANET model to accurately predict observed storage tank data given an anticipated weekly demand pattern, and (b) assist in making lowest-cost pump scheduling decisions for a particular operating period when given a forecasted system demand. This methodology is applied to create a hydraulic model of an actual WDS from observed data and optimize its pump scheduling. In this work, the physical parameters of the simplified model must be calibrated to observed data before pump scheduling optimization can be performed. Because EPANET is a free and flexible WDS-modeling tool developed for the United States Environmental Protection Agency (Rossman 2000), the creation of a functional simplified hydraulic network model is straightforward. The optimization methodology should be capable of scaling to a higher-resolution calibrated model; however, the quality of the calibrated model will be limited by both the accuracy of the provided data and its ability to fully describe the system's behavior. Although optimization of individual pumps can be accomplished through physical improvements to the pumps, such as polishing the pump barrel, trimming the impeller, or even replacing the pump with a more efficient one, this work focuses only on operational improvements.

Calibration and optimization functions are carried out in MATLAB in this work; however, these functions could also be adapted to other, more ubiquitous programming languages such as Java, Python, or Visual Basic.

3.2. Model Formulation

In order to provide real-world data for this research, a certain water utility has graciously provided two years of WDS SCADA (supervisory control and data acquisition) records, recorded in 15-minute increments. This WDS is classified as a “Large” utility by U.S. Environmental Protection Agency standards (population between 10,001 and 100,000) with an average day water demand between 5,000 and 10,000 gpm and a maximum day water demand equal to 1.75 times average day demand in the supplied data. Warm, dry summers are characteristic of the city’s subtropical climate, and extended periods of drought have resulted in peak hourly demands as high as 3.75 times the average day demand flow rate. (At the utility’s request, data that can be used to identify it have been anonymized.) One of the two year-long periods, 2008-2009, was unusually dry and thus its system demand is large and highly variable; the other year-long period, 2006-2007, was relatively wet and thus its system demand is smaller and more constant throughout the day.

In broad terms, the WDS itself consists of a well field (six pumps), a low-head pump station (four pumps), a high-service pump station located several miles away from the low-head pump station (four pumps), and two elevated storage tanks. A schematic

showing the developed EPANET model for this WDS can be seen in Figure 3-1 below.

Refer to Appendix A for the full INP-file formulation of the EPANET model.

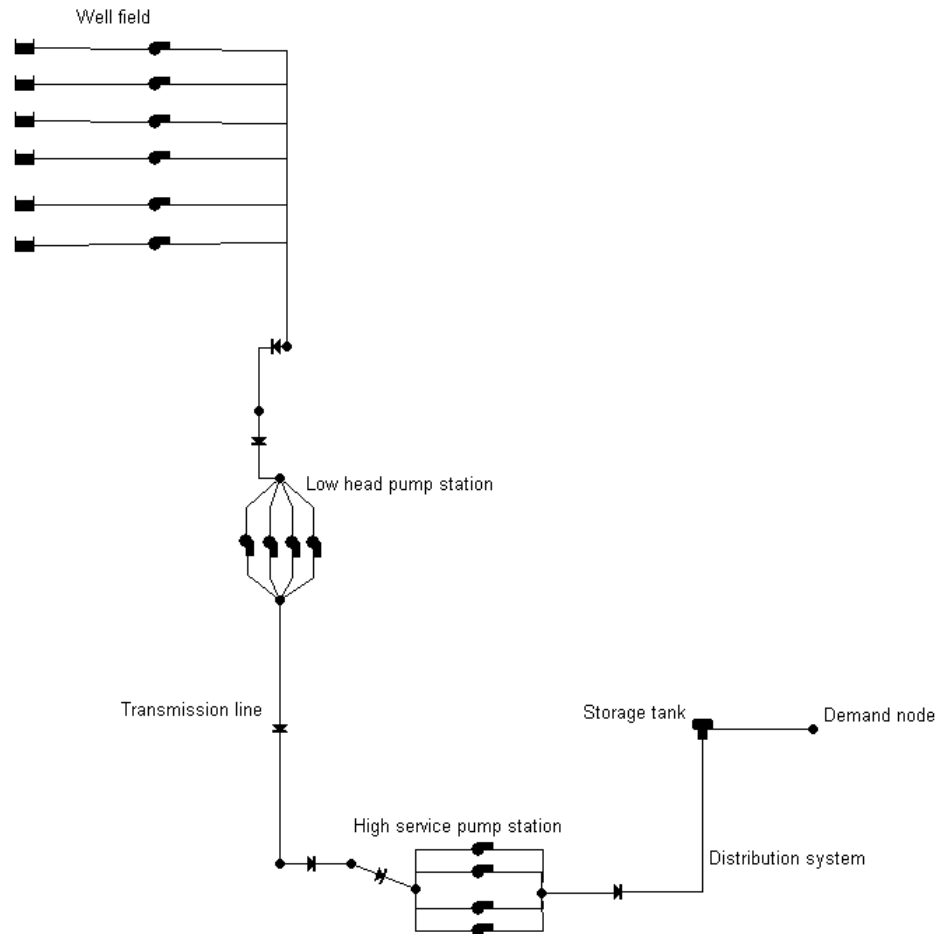


Figure 3-1. Schematic of EPANET model

The provided SCADA records consist of aggregated flow out from the high service pump station and aggregated flow in and out of both storage tanks together. See Figure 3-2 and Figure 3-3 below for a week-long sample of these records. No time

series data of elevated storage volume, elevated storage head, aquifer level, individual pump status, or individual pump operating policy was available. These physical and operational parameters of the model were estimated or calibrated using a genetic algorithm, as detailed in Section 3.3: EPANET Model Calibration.

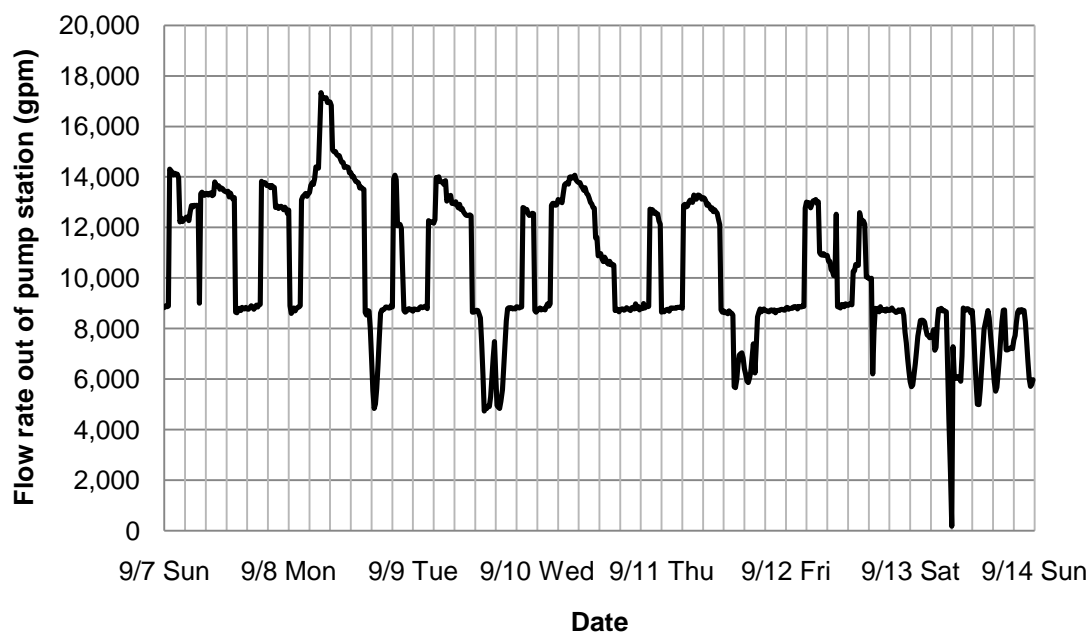


Figure 3-2. Raw incremental HSP station flow rates out, 09/07/08 – 09/13/08

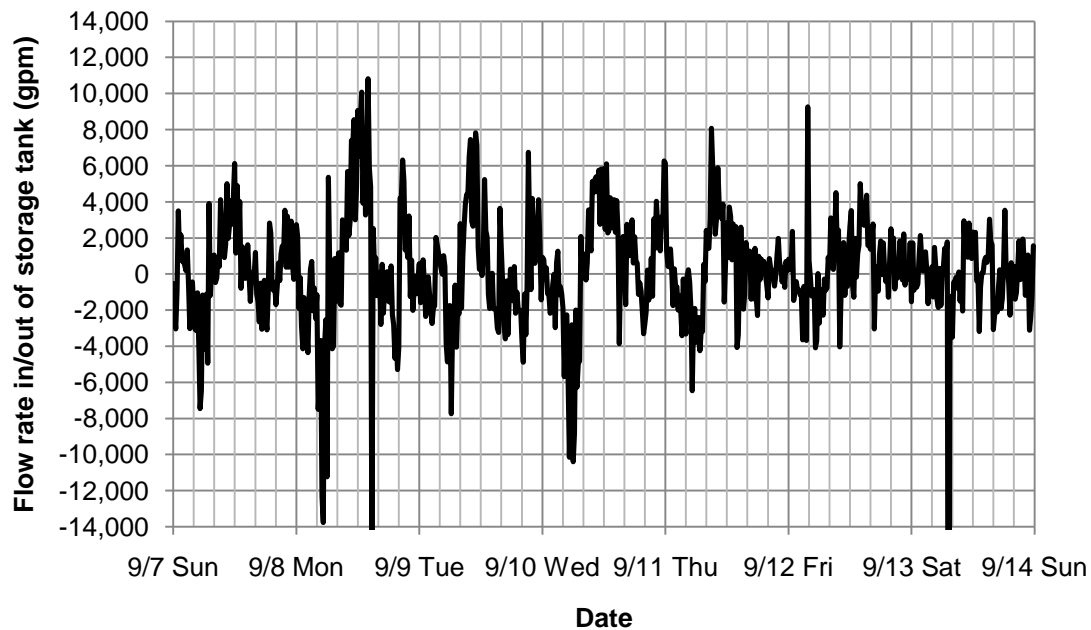


Figure 3-3. Raw incremental flow rate into aggregate storage, 09/07/08–09/13/08

These incremental flow rates formed the basis for calculating a cumulative time series of elevated storage volume. This cumulative storage consistently trended upward, and the difference between calculated minimum and maximum volumes in storage well exceeded the actual maximum storage. One possible explanation of this is that the storage inflow/outflow data recorded by the SCADA system is biased toward inflow at a rate ranging from 26,000 to 465,000 gallons per day, or 18 to 322 gpm. Leaks of this magnitude at some point between the SCADA sensor and the inflow point to the elevated storage tanks may also be a factor. Figure 3-4 and Figure 3-5 below depict the gradually-increasing cumulative storage as calculated from the raw SCADA information.

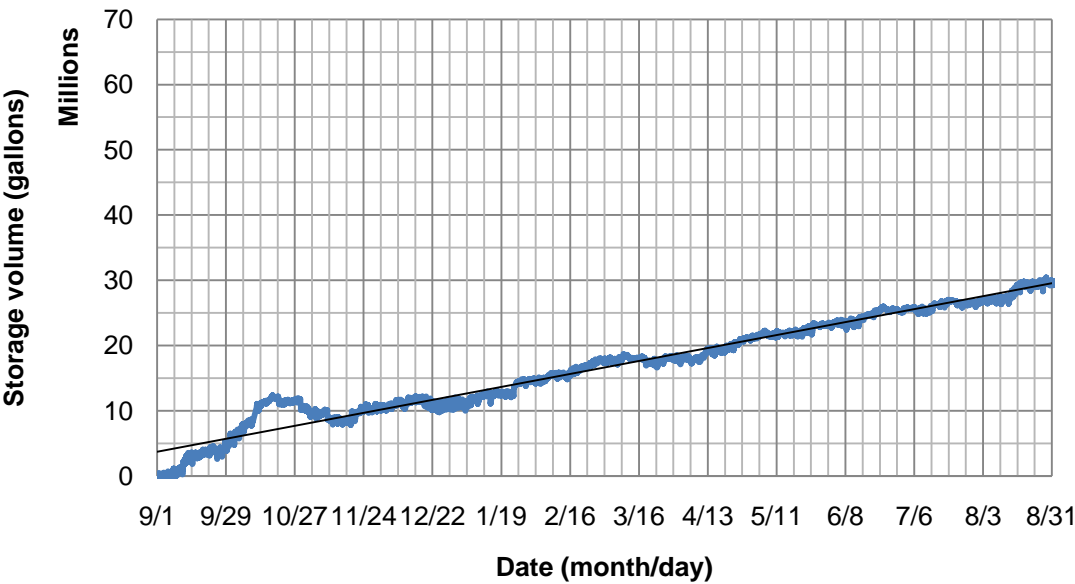


Figure 3-4. Aggregate storage volume from raw SCADA data, wet season

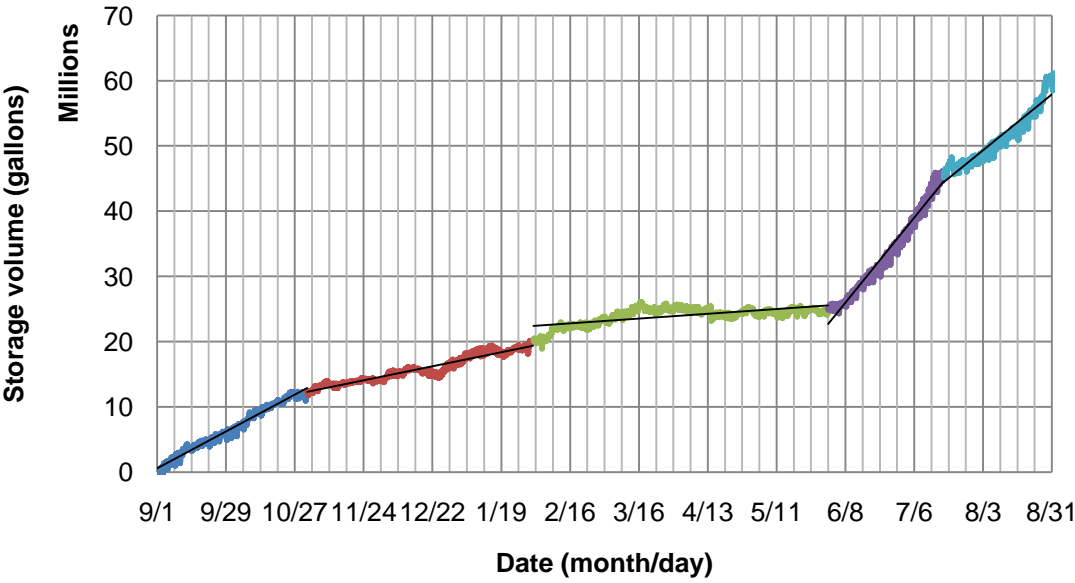


Figure 3-5. Aggregate storage volume from raw SCADA data, dry season

In actuality, elevated storage volume is expected to oscillate around some average value over a long period of weeks or months. In order to correct this upward trending of cumulative tank volume, the SCADA incremental inflow/outflow values to storage were “detrended” by varying rates to produce the expected oscillation of storage volume. During the wet season, the storage volume appeared to accumulate at a relatively constant rate: refer to the slope of the single trendline in Figure 3-4 above. Normal storage oscillation during the wet season was thus achieved by reducing the storage inflow values and increasing the storage outflow values by the slope of this trendline, that is, 46 gpm. During the dry season, however, the storage volume appeared to accumulate at different rates during different periods of the year: refer to the slopes of the five trendlines in Figure 3-5 above. Inflow and outflow values were thus adjusted between the months of September to October, November to January, February to May, June to July, and July to August by rates of 140, 53, 18, 323, and 212 gpm, respectively.

Because the SCADA system recorded only changes in volume and not absolute volume, it was necessary to make an assumption of the volume in storage at some point across the record period. Correspondence with operators of the WDS indicated that the minimum volume in storage necessary for fire protection is 25% of total elevated system volume. To normalize the detrended storage volumes, it was assumed that the tanks had stored exactly this minimum volume at one time during the two-year record period. This detrending and normalization process yielded the final aggregate storage volume series used for model calibration and optimization. These detrended data series are reproduced in Figure 3-6 and Figure 3-7 below.

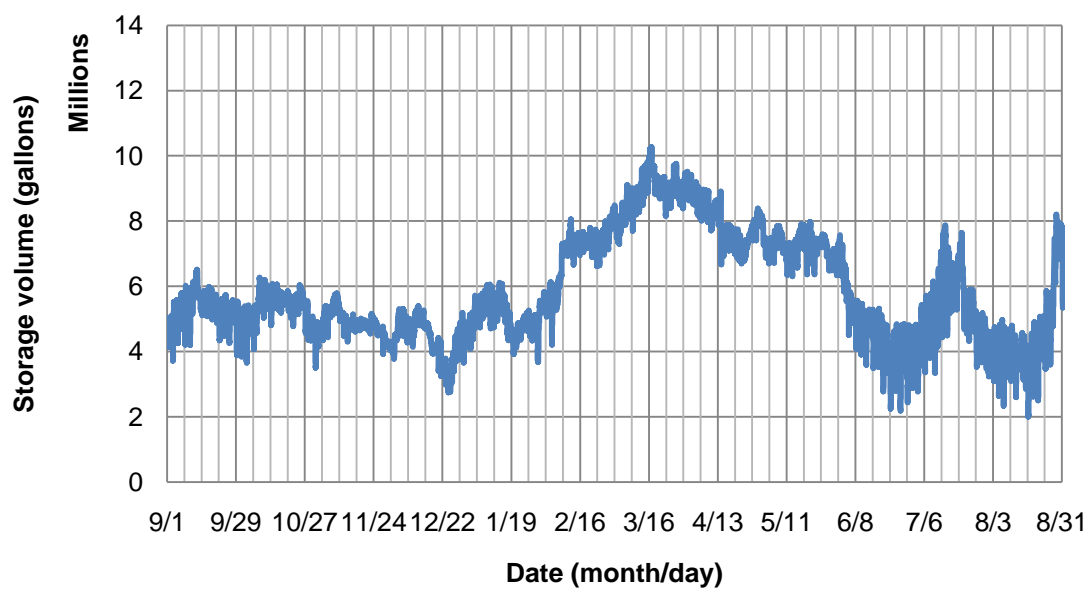


Figure 3-6. Detrended aggregate storage volume, 2006-2007 (wet season)

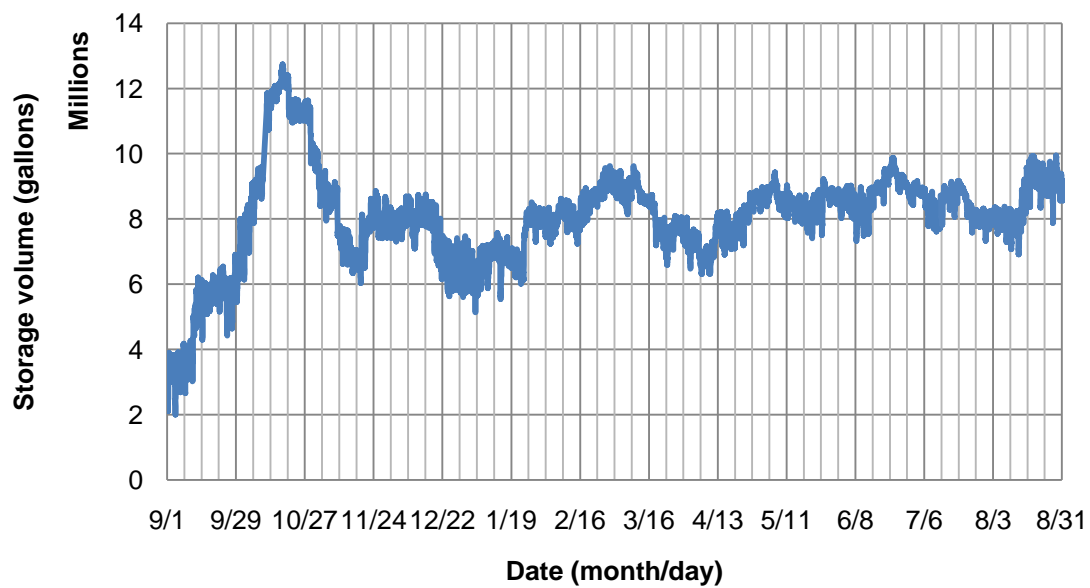


Figure 3-7. Detrended aggregate storage volume, 2008-2009 (dry season)

The total system demand at any particular time, that is, the rate of water use, can be calculated using conservation of mass. This demand is equal to the difference between the HSP flow rate recorded by the SCADA system and the change in detrended storage volume. A set of 104 week-long demand patterns was generated from the data. From these demand patterns, five were selected as representing the full range of the WDS's behavior. These five selected scenarios, whose behavior is detailed in the figures below, describe extreme conditions seen in dry or hot periods (i.e. "2009peak" and "2009peak2"), low-demand wet or cold periods (i.e. "2007flat"), and transitions between these two characteristic extreme periods (i.e. "2009rise" and "2008fall").

Time-series plots of the total system demand (Figure 3-8 through Figure 3-12) are provided for each of the five selected scenarios on pages 24 to 26. Time-series plots of aggregate storage volumes (Figure 3-13 through Figure 3-17) are also provided for these scenarios.

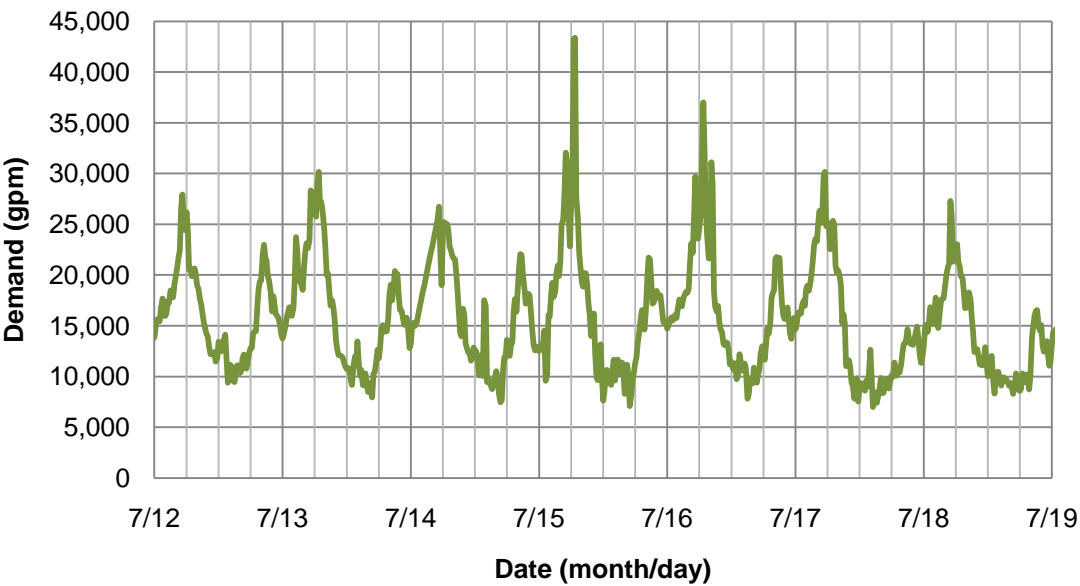


Figure 3-8. Total system demand for Scenario 1 (2009peak2)

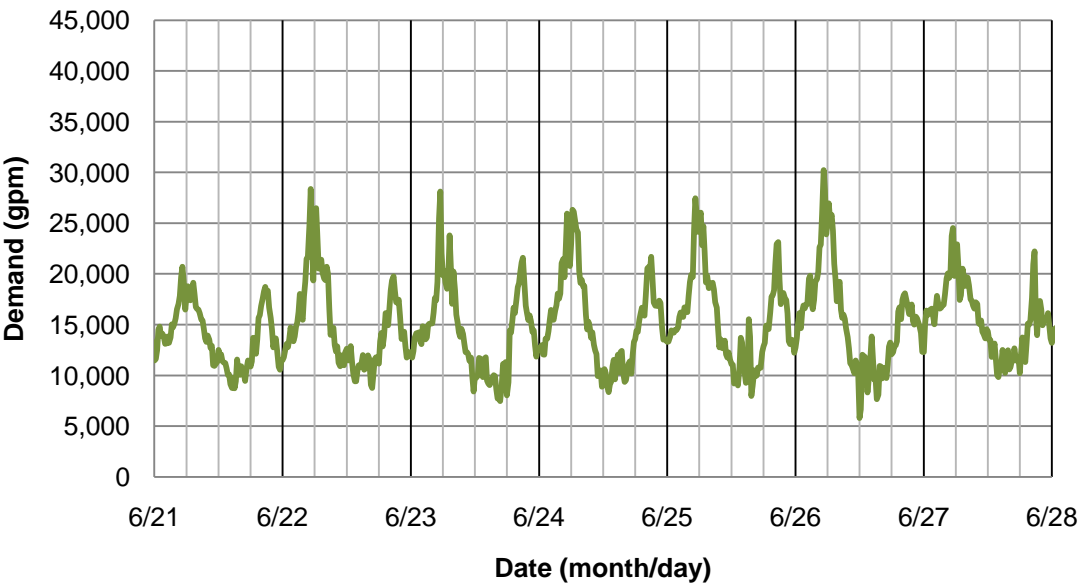


Figure 3-9. Total system demand for Scenario 2 (2009peak)

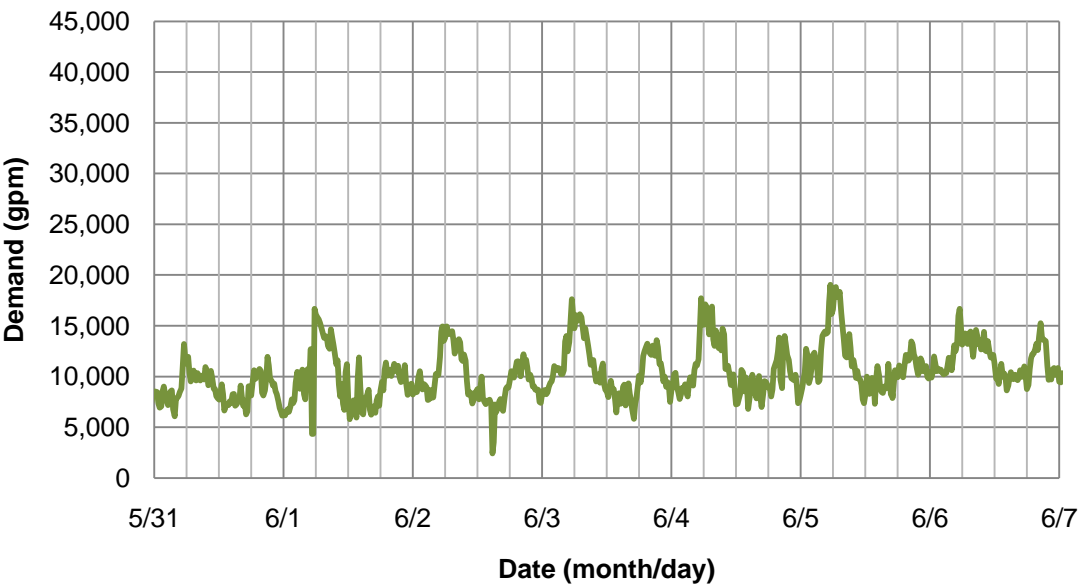


Figure 3-10. Total system demand for Scenario 3 (2009rise)

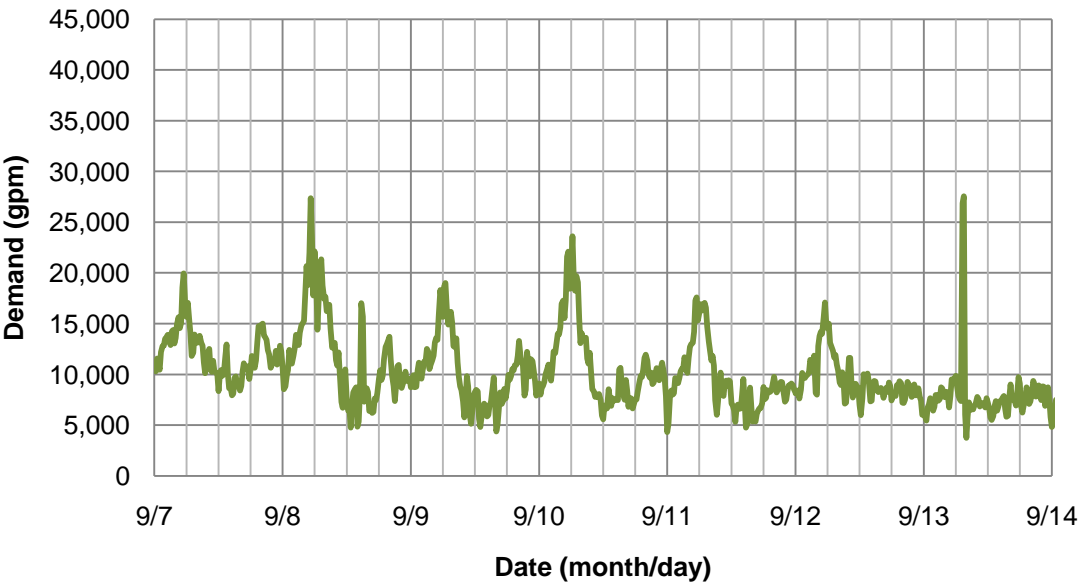


Figure 3-11. Total system demand for Scenario 4 (2008fall)

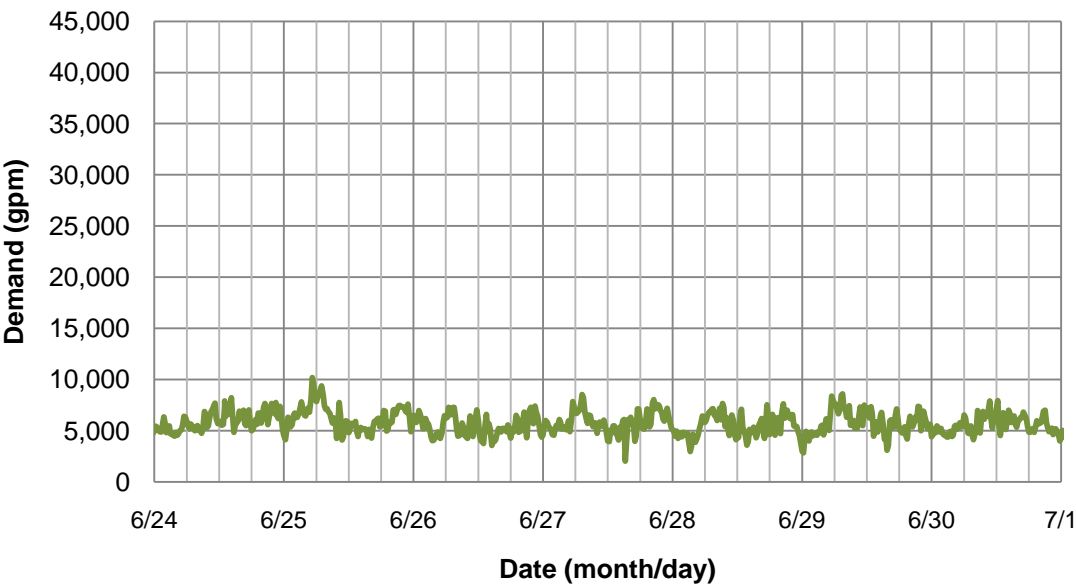


Figure 3-12. Total system demand for Scenario 5 (2007flat)

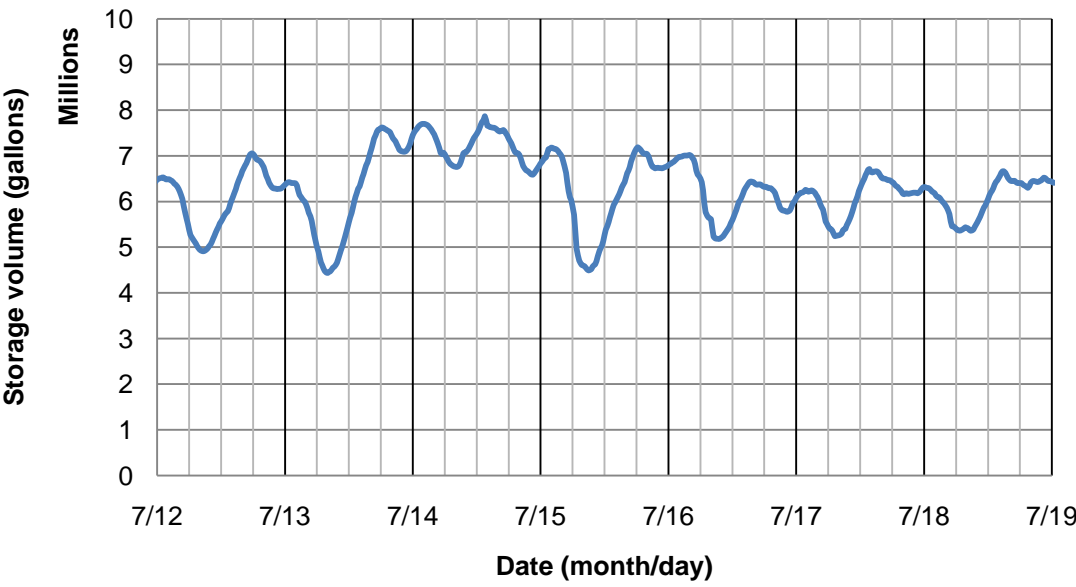


Figure 3-13. Tank storage volume for Scenario 1 (2009peak2)

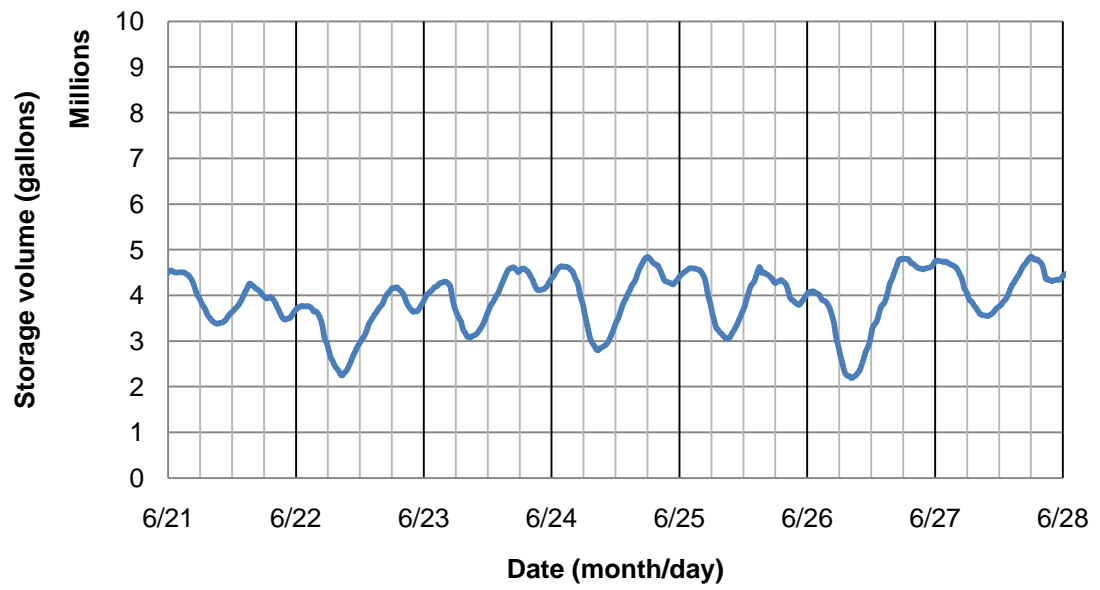


Figure 3-14. Tank storage volume for Scenario 2 (2009peak)

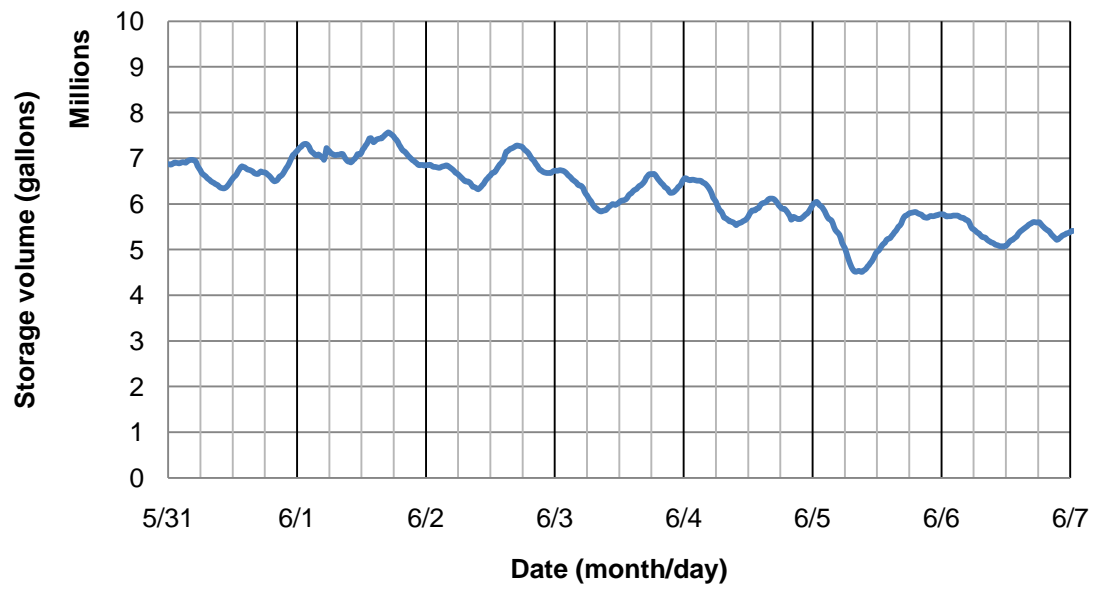


Figure 3-15. Tank storage volume for Scenario 3 (2009rise)

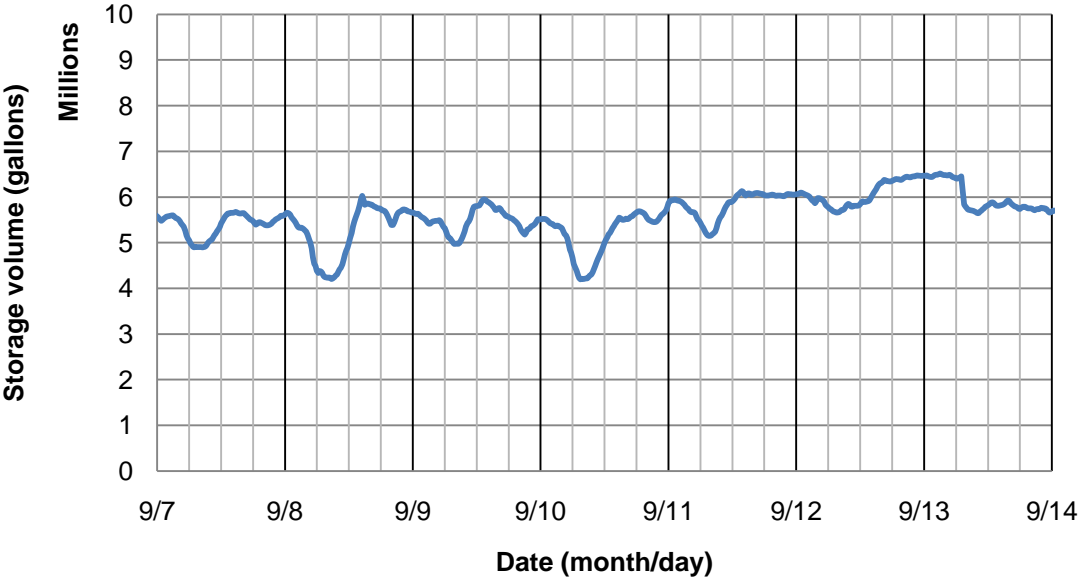


Figure 3-16. Tank storage volume for Scenario 4 (2008fall)

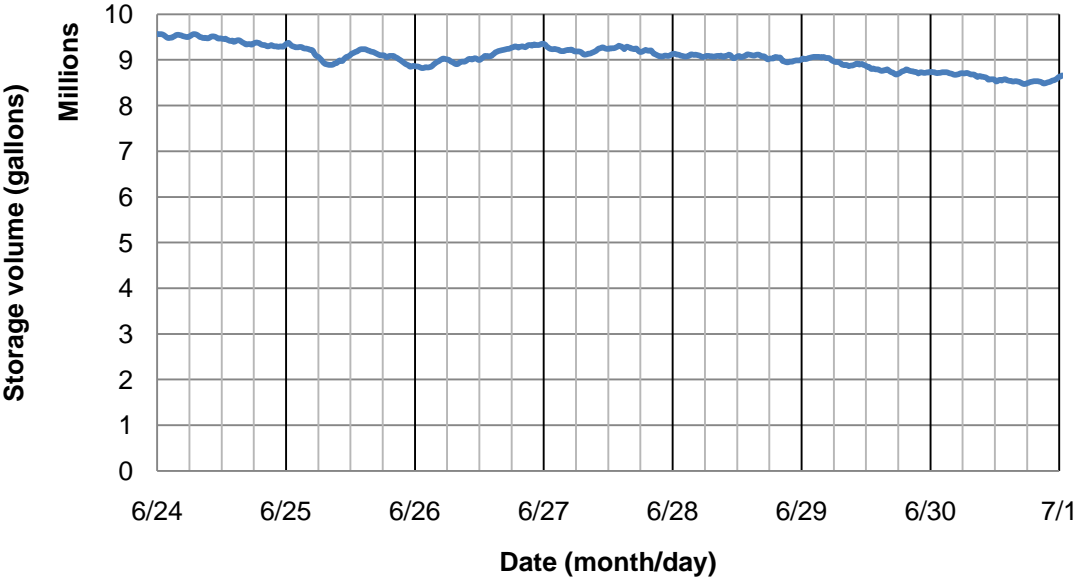


Figure 3-17. Tank storage volume for Scenario 5 (2007flat)

The EPANET model treats the WDS storage as a single elevated tank. In order to model the effects of an elevated tank on the system, EPANET requires the user to define some sort of relationship between storage volume and hydraulic head. This is commonly accomplished by assuming a cylindrical tank shape and defining its diameter and minimum elevation. Thus, for a given storage volume at a certain time step, EPANET can easily calculate hydraulic head at in the WDS at the tank connection.

Actual dimensions of the storage tanks were unavailable. One possible approach to modeling a tank of unknown dimensions would be to make both tank diameter and tank minimum elevation into decision variables to be adjusted in the Model Calibration stage; however, this would increase the scope of the calibration problem. In an effort to shrink the expected size of the decision space and thus reduce computation time, a single cylindrical tank with a diameter of 150 feet was fixed in the Model Formulation stage. The minimum tank elevation, however, was calculated during the Model Calibration stage. Because the aquifer water surface elevation (WSEL) at the well pumps was also an unknown physical parameter, the absolute difference in elevation between the minimum tank elevation and the aquifer WSEL was chosen as a decision variable, dz , in the Model Calibration stage.

The water utility also provided pump curves and energy efficiency curves for three of the six well pumps, two of the four low-head transmission pumps, and all four high-service distribution pumps. To simplify model simulation, only two of the three provided well pump curves were used. The first of these two curves is referred to in the model as “Well3” and has a cutoff head of 560 ft. The second is “Well5” and has a

cutoff head of 505 ft. Each of these two curves was assigned to three of the six WPs in the model. The two low-head transmission pump curves that were provided are virtually identical, and so only one curve, “LHP,” was used in the model for all four LHPs. Of the four high-service distribution pump curves that were provided, two are equipped with 16.25” impellers while the other two are equipped with 15.75” impellers. To simplify model simulation, only two of the four high-service distribution pump curves were used. The first of these two curves is referred to in the model as “HSP1” and has a cutoff head of 277 ft. The second is “HSP2” and has a cutoff head of 250 ft. Each of these two curves was assigned to two of the four HSPs in the model. All pump curves discussed here are provided below in Figure 3-18 through Figure 3-22.

Please note that the efficiency values discussed throughout this work are hydraulic efficiency values which describe the rate of energy transfer between the pump impeller and the water itself. Mechanical efficiency, which describes the rate of energy transfer between the electric motor and the pump impeller, is not considered.

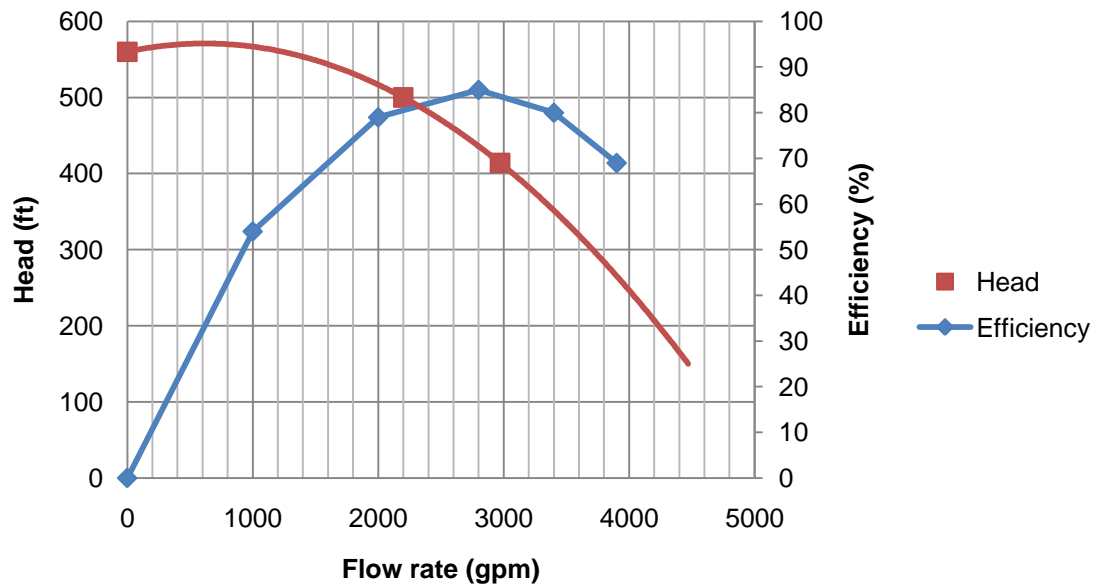


Figure 3-18. Characteristic and efficiency curves for pump Well3

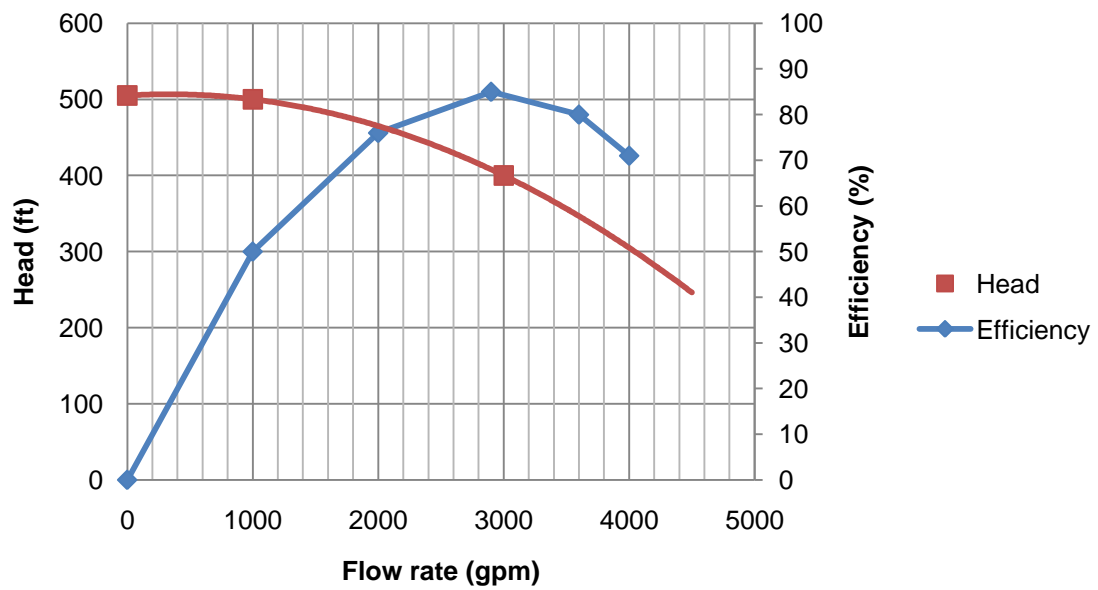


Figure 3-19. Characteristic and efficiency curves for pump Well5

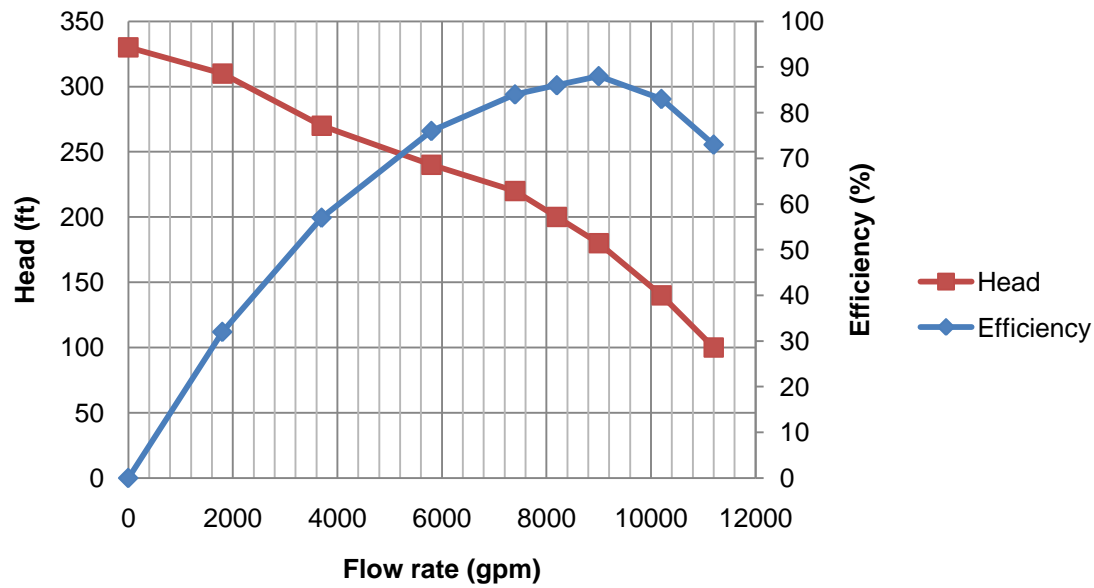


Figure 3-20. Characteristic and efficiency curves for pump LHP

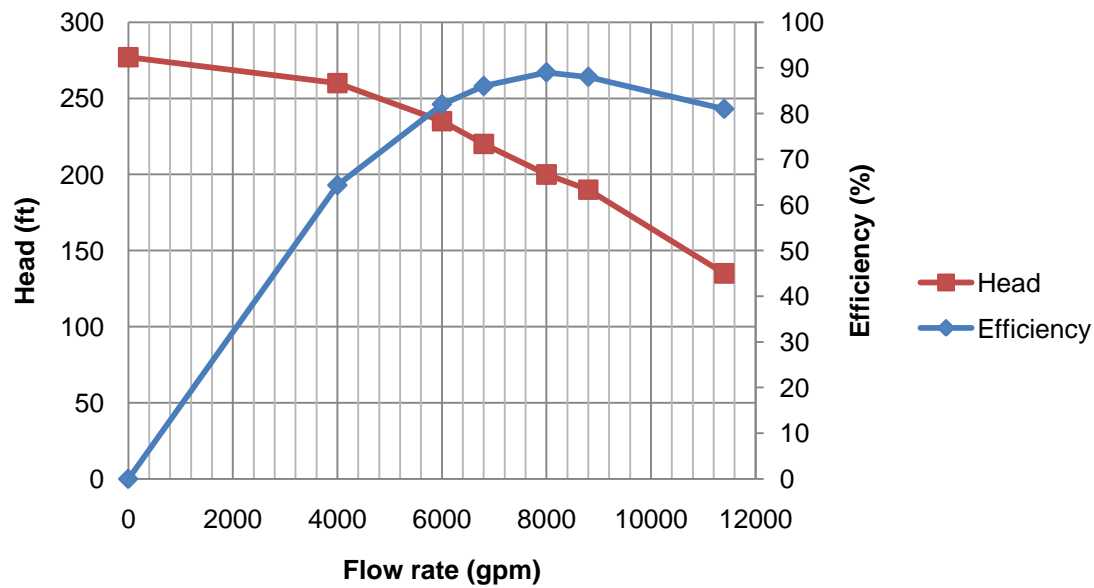


Figure 3-21. Characteristic and efficiency curves for pump HSP1

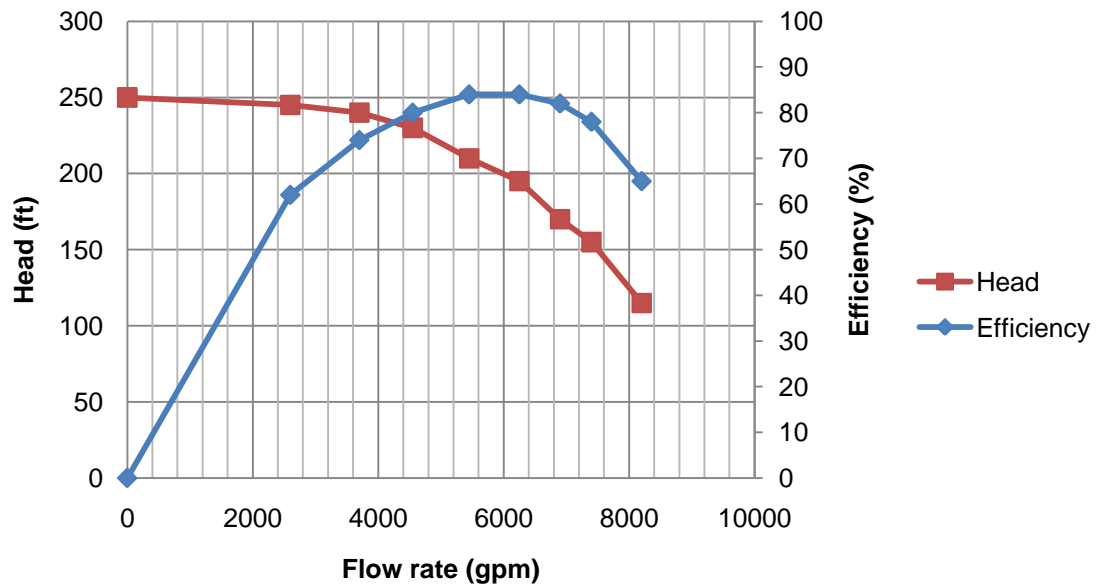


Figure 3-22. Characteristic and efficiency curves for pump HSP2

Most pumps in this system are constant-speed pumps, but one of the low-head pumps and two of the high-service pumps are equipped with variable frequency drives (VFD). While the constant-speed pumps have only one best-efficiency operating point, a VFD can vary the speed of the pump motor continuously, allowing an operator to choose from a wide range of possible operating points without sacrificing efficiency. That is, by lowering the speed of a VFD pump, an operator can pump a lower flow rate at a lower pressure while remaining at the best-efficiency operating point. The effect of the VFDs can be accounted for in EPANET by specifying a speed parameter describing the fraction of its maximum speed. The pump curve is then adjusted such that flow rate values are multiplied by the speed and hydraulic head values are multiplied by the

square of the speed. The efficiency curve is adjusted such that flow rate values are multiplied by the speed and efficiency values remain the same.

For example, consider a VFD-equipped “HSP2” pump that imparts 115 ft of hydraulic head. At full speed, this pump would operate at a flow rate of 8200 gpm for an efficiency of only 65%. Unfortunately, the best-efficiency point (84%) is 195 ft of head at 6200 gpm. If the operator were instead to run the pump at a speed of 75%, the pump would be able to impart the same 115 ft of hydraulic head to a flow rate of about 4600 gpm, resulting in a maximum efficiency of 84%.

Table 3-1 below summarizes the names of all pumps in the system, the pump curves each one uses, and whether each one’s speed can be adjusted through a VFD.

Table 3-1. Pump summary table.

Pump Name	Pump Curves	VFD-equipped
WellPump1	Well3	No
WellPump2	Well5	No
WellPump3	Well3	No
WellPump4	Well5	No
WellPump5	Well3	No
WellPump6	Well5	No
LHP1	LHP	Yes
LHP2	LHP	No
LHP3	LHP	No
LHP4	LHP	No
HSP1	HSP1	Yes
HSP2	HSP2	Yes
HSP3	HSP1	No
HSP4	HSP2	No

A simplified hydraulic model of the WDS was created using EPANET version 2.00.12a (Rossman 2000). Please refer back to Figure 3-1 on page 17 for a schematic of the model, which comprises four main parts modeling the well field, the low-head pump station, the high-service pump station, and the distribution system. The well field includes six reservoir nodes connected directly to six individual well pumps. These are connected by one transmission link to the four low-head pumps, which are in turn connected by one transmission link to the four high-service pumps. Because the focus of this thesis is the energy use at each pump station and not the precise characteristics of the distribution system itself, the entire distribution system after the high-service pump station is modeled as two distribution links separated by a combination storage tank and demand node. All energy loss due to friction in the entire distribution system is assumed to occur within these two links.

See Appendix A for the full EPANET model in plain-text INP format.

The EPANET model is simulated within MATLAB R2010a, which serves as a framework for the model to be executed iteratively. The function `cityWdsRun.m`, reproduced in Appendix B, was designed to open the EPANET INP file of the WDS, initialize it with certain input parameters, and solve for hydraulic unknowns such as flow rates, pressures, tank levels, and pump energy usage. It requires a copy of `epanet2.dll`, known as the EPANET Toolbox (Rossman 2000), to access EPANET's functions for performing hydraulic analysis.

Input parameters to the model are denoted as the “inpt” vector and include the length, roughness, and diameter for four major pipe groups (WPs to LHPs, LHPs to HSPs, HSPs to the storage tank, and the storage tank to the demand node) and an initial storage tank level set at the beginning of the simulation (dz). Another set of input parameters, denoted as the “pumpParam” vector, include four numbers describing the initial state of all pumps, that is, on/off status for constant-speed pumps and a fraction of maximum speed for VFD-equipped pumps. Because actual pump operation data is not available, these initial pump statuses are assumed to persist throughout the duration of the simulation.

To carry out a particular week-long simulation, `cityWdsRun.m` first opens the EPANET INP file, and then loads the proper demand pattern, physical system parameters, and initial pump parameters specified by the user. The function proceeds to carry out a seven-day simulation using a hydraulic time step of one hour, and outputs a time-series of head in the elevated storage tank as well as a time-series of pump statistics, including energy expended in kilowatt-hours.

During model calibration, `cityWdsRun.m` is able to compare the simulated time series of head in the elevated storage tank with the actual time series of head observed by the SCADA system. This results in another output variable, “`tankError`,” which is the sum of squared errors between the two time series. This variable is to be minimized by a genetic algorithm through the functions `calibrateGA.m` and `cityWdsCal.m` as detailed in the next section, 3.3 – EPANET Model Calibration.

During model optimization, `cityWdsRun.m` is also able to receive additional input variables from `optimizeGA.m` and `cityWdsOpt.m` describing time series of forecasted demands (‘`optForecast`’), time series of desired pumped flow (“`optPumpFlow`”), and beginning and desired ending tank levels (“`optTankLvls`”). At specified time intervals, the function determines the most efficient pump combination by comparing Q , the user-specified desired pump flow, and dz , the elevation difference between the aquifer and the tank level, with a table of predetermined optimal pump combinations for every possible Q - dz combination. This process is detailed in Section 3.4 – EPANET Model Optimization.

All relevant MATLAB functions used are reproduced in Appendix B.

3.3. EPANET Model Calibration

3.3.1. Genetic Algorithm Background

Consider an objective function f with a vector of input parameters x . The value of this function f varies with each individual value, or decision variable, contained in the vector x . The purpose of optimization is to locate a set of decision variables which returns a desired value of f – usually the highest or lowest possible value. A common example is a set of two decision variables describing latitude and longitude. An objective function returning the ground surface elevation can be evaluated for any given set of these decision variables. By repeatedly evaluating the function using random values of latitude and longitude, discarding the lower values and keeping the highest ones, a modeler may eventually be able to find the highest mountain peak in a certain decision space. However, he cannot be completely confident that his solution is at a global maximum without fully evaluating the continuous elevation function across the entire decision space. In this example, there are only two decision variables; the decision space is two-dimensional. A three-dimensional optimization problem may entail, for example, finding the coldest point in a large three-dimensional space. As the dimensionality of the decision space increases, the objective function becomes more complex, making this process of searching for global maxima or minima more difficult. Even if a good solution is found, the confidence of having found a global optimum is decreased, and exhaustive evaluation of a function with many decision variables is infeasible. If one has any hope of finding even a local optimum, one must implement an

algorithm that explores the decision space broadly at first, then more narrowly as good solutions are found. The goal is to find the optimum point while limiting the number of function evaluations.

A genetic algorithm (GA) is a commonly-used approach to generating solutions to a problem with many decision variables. It is a type of evolutionary algorithm that minimizes any given non-linear function through a mathematical process based on the principles of natural selection. Its methods of population generation, selection, crossover, and mutation are flexible and can be adapted to fit a variety of problem types.

To minimize an objective function, the GA first randomly generates a large population of solution vectors within acceptable lower and upper bounds. Each vector is then evaluated by the function to be minimized. Then, the solution vectors are sorted according to their corresponding function values from smallest to largest; the two smallest function values are considered most fit and are replicated into the next generation. At this point, the next generation of vectors is generated by means of crossover or mutation. The crossover operator takes two parent vectors, selected randomly from the current generation, and generates a child vector for the next generation by randomly selecting single variables from each parent vector. The mutation operator takes each individual vector of the population and changes one of its variables to a new, random value. The rates of crossover and mutation are specified by the modeler.

3.3.2. Calibration Procedure

Refer to Figure 3-23 below to see an example set of data from Scenario 3 (2009rise). The figure shows the system demand from Figure 3-10 plotted against the tank storage volume from Figure 3-15. The overall goal of the calibration procedure is to come up with a set of physical and operational model parameters such that the model can accurately predict the output tank storage volume (denoted by the blue line) when given the input system demand (denoted by the green line). Note that this scenario shows a gradual increase in system demand across the week, and that it corresponds with a gradual draining of the volume in storage.

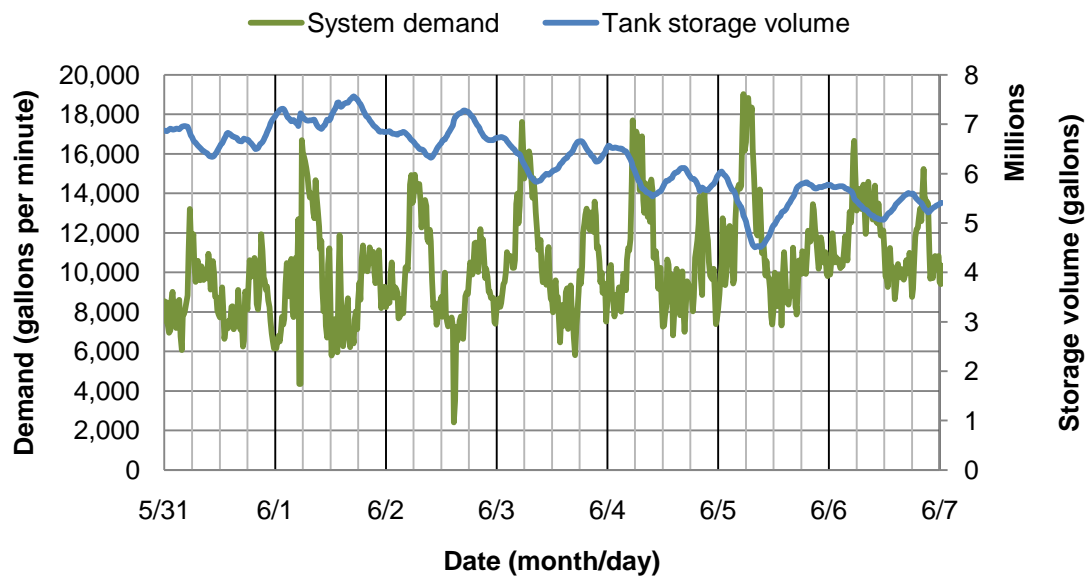


Figure 3-23. Total system demand and tank storage volume for Scenario 3 (2009rise)

A general-purpose GA included with MATLAB was used to calibrate physical parameters of the EPANET model to minimize the sum of squared errors between the tank level simulated by EPANET and the SCADA-observed tank level for the five demand scenarios mentioned earlier (the two summer periods 2009peak2 and 2009peak, the transition periods 2009rise and 2008fall, and the winter period 2007flat). By minimizing the objective function of the difference between simulated and observed tank levels, the calibration process provides a set of parameters that cause the model to approximate the behavior of the actual WDS. (Note that the formulation presented here is specific to this work; alternate or modified optimization algorithms could be explored to develop better solutions more quickly.)

Additional penalty values to the objective function were assessed in order to prohibit infeasible solutions from remaining in the population, regardless of tank error value. These penalty values were applied to the objective function when certain error codes were returned by EPANET – that is, when EPANET calculated negative pressure at a node, when flow through a pump was greater than the largest flow value on its characteristic curve, or when head across a pump was greater than its cutoff head. The most serious penalty was applied when the decision variables applied to the model made it unstable and/or prevented EPANET from converging to a hydraulic solution; in this case, execution of the model was instantly stopped and an extremely large penalty value was applied. This penalty formulation helped the calibration procedure to run more quickly, as completely infeasible solutions can be penalized and effectively discarded at the instant a serious error is encountered.

For initial calibration operations, 70% of every generation was generated by the crossover operator. The remaining 30% were generated by mutation. The mutation operator took each individual vector and changed one variable to a new, uniformly distributed random value with a probability of 20%. The fitness of each individual in the new generation is evaluated, and the process repeats. A population size of 100 and a generation size of 30 was used, and the calibration procedure was executed in three consecutive trials before choosing an optimal solution from those trials.

MATLAB comes packaged with an optimization toolbox, including a general-purpose genetic algorithm function, `ga.m`, for minimizing non-linear functions. All GA parameters described in the previous paragraph were specified using MATLAB's built-in option-setting tool, `gaoptimset.m`.

Initially, 33 decision variables were used in calibration of the EPANET model. The first group of 13 variables included 12 variables describing the pipe diameter, Hazen-Williams roughness, and length for four separate pipe groups and one variable describing the change in elevation between the aquifer surface and the bottom of the storage tank. These values remain the same during each of the five observed demand scenarios. The next 20 variables include, for each of the five observed demand scenarios, four numbers describing the number of WPs on, the number of constant speed LHPs on plus the percent speed of the VFD-equipped LHP, the number of constant-speed HSPs on, and the percent speed of the two VFD-equipped HSPs, respectively. Thus, the 33 variables together represent a unique set of parameters for the model that

sufficiently describe the WDS' behavior during five separate observed scenarios. Refer to Table 3-2 below for a typical set of these 33 decision variables.

Table 3-2. Example set of 33 decision variables for model calibration.

<i>Pipe Properties (ClearWellIn: WP to LHP)</i>			
<u>Diameter, in</u>	<u>Length, ft</u>	<u>H-W Roughness</u>	
36	1,000	100	
<i>Pipe Properties (Transmission: LHP to HSP)</i>			
36	60,000	100	
<i>Pipe Properties (Distribution: HSP to Storage Tank)</i>			
36	20,000	100	
<i>Pipe Properties (Distribution2: Storage Tank to Demand Node)</i>			
36	10,000	100	
 <i>Tank Elev above Wells, ft</i>			
450			
 <i>Pump Parameters (2009peak2)</i>			
<u>Initial # of WPs On</u>	<u>Initial # of LHPs On + Speed of VFD LHP</u>	<u>Initial # of Constant-Speed HSPs On</u>	<u>Initial speed of VFD HSPs</u>
6	3.8	2	0.9
<i>Pump Parameters (2009peak)</i>			
6	3.8	2	0.9
<i>Pump Parameters (2007flat)</i>			
3	2.0	1	0.8
<i>Pump Parameters (2009rise)</i>			
4	2.0	1	0.8
<i>Pump Parameters (2008fall)</i>			
4	1.7	1	0.7

The main problem with approaching the calibration with a large number of decision variables is that it creates an extremely large, complex decision space that is difficult to explore for feasible solutions. Another major issue is the GA's unawareness of the relative importance of each decision variable with respect to each demand scenario. For example, a candidate solution that returns a high objective value may be an excellent fit for the first four scenarios, but a terrible fit for the fifth one. In this situation, a rational modeler would leave the pipe and tank parameters alone and instead focus on tweaking the pump status parameters specific to that fifth scenario, i.e. the last four decision variables. The GA lacks this sophistication and instead considers all 33 variables to be just as valid a choice for mutation or crossover as those few pump status parameters. This means that the GA is more likely to change a variable affecting all scenarios, e.g. a pipe diameter, thus greatly reducing the accuracy of the first four modeled scenarios and throwing away what was actually very close to a good solution. This approach required far too many function evaluations to return anything close to a feasible solution, and was not able to do so reliably.

To alleviate this problem, the final calibration procedure was separated into two steps:

1. The first step would calibrate the 13 physical parameters (pipe properties and elevation difference) five times – once for each individual demand scenario.

This allows the procedure to use only one set of four pump parameters per calibration run, rather than five sets of four pump parameters, bringing the total number of decision variables for the first step down from 33 to 17.

2. The second step would calibrate the full 33-decision-variable model of all scenarios a total of five times, and would choose one best solution from these five calibration runs. Each of the five calibration runs used an initial population assembled from the calibrated solutions from step 1 as a starting point; that is, the GA would fill the initial 33-decision-variable population entirely with members identical to the first step's calibrated solutions rather than with randomly generated ones. Thus, the second step fine-tunes the initial solution provided by the first step.

Each of the five calibration runs in the second step assembled its initial population of 33 decision variables using a combination of the corresponding set of 13 physical parameters calibrated in the first step and all 20 pump parameters from the first step. That is, all five second-step calibration runs used the same initial 20 pump parameters; the only parameters that changed with each calibration run were the physical parameters. Because the physical parameters calibrated in the first step affect the overall fitness and stability of the model to a greater degree than the individual scenarios' pump configuration parameters, focusing on their calibration first leads to a reasonable solution more quickly and reliably.

See Table 3-3 below for an example of the 17 decision variables used in Step 1 of the model calibration. Step 2 of the model calibration used the 33 decision variables described previously in Table 3-2.

Table 3-3. Example set of 17 decision variables for model calibration.

<i>Pipe Properties (ClearWellIn: WP to LHP)</i>		
<u>Diameter, in</u>	<u>Length, ft</u>	<u>H-W Roughness</u>
36	1,000	100
<i>Pipe Properties (Transmission: LHP to HSP)</i>		
36	60,000	100
<i>Pipe Properties (Distribution: HSP to Storage Tank)</i>		
36	20,000	100
<i>Pipe Properties (Distribution2: Storage Tank to Demand Node)</i>		
36	10,000	100

<i>Tank Elev above Wells, ft</i>
450

<i>Pump Parameters (one individual scenario)</i>			
<u>Initial # of WPs On</u>	<u>Initial # of LHPs On + Speed of VFD LHP</u>	<u>Initial # of Constant-Speed HSPs On</u>	<u>Initial speed of VFD HSPs</u>
5	3.0	2	1.0

3.3.3. Uncertainty Analysis

After calibration is performed and a well-performing set of EPANET model parameters is decided upon, these parameters can be permanently applied to the model. The model can now be used for simulation of any given combination of pump combinations and demands. However, at this point the robustness of the model is somewhat of a mystery. The model could be quite sensitive to small variations in each of the calibrated parameters. If, for example, the diameter of one pipe were changed from 36 inches to 35 inches, would the model completely lose its ability to predict the behavior of the actual WDS? If so, the model would be considered very sensitive, and such sensitivity could help explain poor performance under certain conditions. If, on the other hand, the EPANET model could accurately predict the behavior of the actual WDS even if all parameters were randomized by 20%, for example, the model would be considered robust and capable of returning precise, consistent results.

Monte Carlo uncertainty analysis is one way of quantifying the robustness of a calibrated model. It entails executing the calibrated model repeatedly while allowing random variations of its parameters within a specified tolerance. The effects of uncertainties of 10%, 20%, 30%, and 40% were measured by executing the calibrated

EPANET model after adjusting each calibrated parameter randomly along a normal distribution such that two standard deviations above or below the mean, i.e. 95.4% of the normal distribution, is within the specified uncertainty percentage. That is, for an uncertainty value of 10%, a Hazen-Williams roughness parameter of 100 would be normally randomized such that there would be a 95.4% chance that the newly generated roughness value would lie between 90 and 110. For each Monte Carlo simulation, each of the 33 decision variables was randomized. Two hundred Monte Carlo simulations were performed for each uncertainty percentage, and all results were saved.

After completion of the Monte Carlo analysis, the minimum, maximum, 25th, 50th, and 75th percentiles of the tank head in feet were plotted for each uncertainty percentage and compared with the observed tank head in feet. Refer to the section on uncertainty analysis on page 64 for these results, as well as discussion of the sensitivity of the calibrated EPANET model.

3.4. EPANET Model Optimization

The ultimate goal of developing a calibrated EPANET model is to develop a pump scheduling optimization function which can prescribe efficient pump settings given a predicted demand. While the goal of calibration was to minimize error between observed and simulated tank levels by changing physical parameters, the goal of optimization is to minimize total power usage by changing pump configurations at specified intervals. Although power cost is not directly addressed in this work, the optimization and simulation routines could someday be adapted to account for diurnal energy rates and to minimize total power cost rather than total power usage. A GA was also implemented for purposes of optimization.

A problem arises when applying the decision variable schema from the calibration procedure directly to the optimization procedure. Suppose that the pump operator has a forecast of the next 24 hours of system demand, and wants to change pump configurations every six hours, i.e. four times per day. Consider the four pump parameter decision variables described in Section 3.3.2 – Calibration Procedure. If the decision variables in the optimization GA were to be formulated using four sets of these four pump parameters, the resulting 16-dimensional decision space would be massively discontinuous. That is, of the millions of possible pump combinations described by those 16 variables, the vast majority are infeasible, either because the combination is extremely inefficient, or because the combination operates one or more pumps at a point above its cutoff head or beyond its maximum flow capacity. To further complicate matters, certain combinations may be inefficient given certain system conditions, e.g.

demands and tank levels, but these combinations may become efficient given different system conditions. Although convergence using a GA or other minimization technique is theoretically possible here, it would be computationally expensive. Early experimentation with this type of formulation resulted in runtimes exceeding 24 hours on an Intel Pentium III processor operating at 3.0 GHz. Since the optimization runtime exceeds the duration of the forecasted demand period, this formulation is entirely impractical for short-term demand forecasting.

3.4.1. Enumeration of Efficient Pump Combinations

To simplify the optimization process, it can be assumed that there exists a small set of pump combinations whose weighted efficiency remains high (70-80%) across a wide range of discrete pumping flow rates. This assumption is valid considering the millions of possible pump combinations available: first, because each pump has a specified efficiency curve with a well-defined best-efficiency point, and second, because three of the fourteen pumps are equipped with VFDs, giving them a continuous range of possible best-efficiency points. Operating on this assumption, all possible pump combination characteristic curves and system head loss curves were evaluated. Intersecting any given pair of characteristic curve and system head loss curve yields one operating point describing the total flow and head imparted by the pumps. Of these operating points, the inefficient ones can be discarded, leaving a set of elite efficient pump combinations across a wide range of flow values.

This numerical evaluation of efficient operating points is carried out by a deterministic function in MATLAB (*enumPumps.m*). First, simple quadratic equations are fitted to each of the five provided pump curves – two WP curves (“W3” and “W5”), one LHP curve (“L”), and two HSP curves (“H1” and “H2”). These curves relate the total flow Q through the pumps in gallons per minute to the total head H imparted to the pumped flow in feet. See Figure 3-18 through Figure 3-22 on pages 31 to 33 for plots of these pump curves.

The head at each point along the curve is evaluated numerically at discrete intervals of 5 gpm. Because one LHP and two HSP curves are equipped with VFDs, the head values of those curves are evaluated at discrete speeds of 20%, 40%, 60%, 80%, and 100%, where pump flow is reduced directly proportional to pump speed and pump head is reduced proportional to the square of pump speed. Next, all possible combinations of these individual pump curves are evaluated by interpolation and simple addition. All WPs are in parallel, as are LHPs with one another and also HSPs with one another. Furthermore, the WP station is in series with the LHP station, which is in series with the HSP station. Flow through pumps in series remains constant while head is added, and head across pumps in parallel remains constant while flow is added. The resulting final characteristic curves for every possible pump combination relate the total flow pumped from the wells (Q) to the head difference between the wells and the water level in the storage tank (dz).

The final computed set of 2,346 characteristic curves is accompanied by a corresponding set of characteristic efficiency curves, obtained by weighting the original

pump efficiency curves according to head (for pumps in parallel) or according to flow (for pump stations in series).

The family of system head loss curves is described by the summation of the Hazen-Williams equation $= \sum(K_{eq} Q^{1.85})$ across pipes in series, where $K_{eq} = 4.727 C^{-1.852} D^{-4.871} L$, Q is the flow through each pipe in gallons per minute, C is the unitless Hazen-Williams roughness, and D is the pipe diameter in feet. All pipe parameters used here are the same parameters resulting from the model calibration as described in Section 3.3.2 above (Calibration Procedure). The values of dz were chosen to range from 630 to 780 feet in 1-foot increments in order to encompass all expected values of dz between the WSELs of the wells and the storage tank in the EPANET model.

Finally, all possible intersections of the discrete pump characteristic curves and the system head loss curves, that is, all possible operating points, are calculated with the assistance of the *mmcurvex.m* function of the Mastering MATLAB 6 Toolbox (Hanselman and Littlefield 2001), and only those operating points that meet a minimum desired efficiency of 60% are saved. The least-efficient operating points within 50 gpm of one another are then discarded. Each saved operating point, then, has associated with it a given change in elevation dz and a given pumped flow Q . This static set of 27,784 efficient operating points (*opPts*) can serve as a lookup table for the optimization procedure. At any time during the simulation, the dz value describing the difference between the aquifer level and the storage tank level can be measured and paired with a desired flow Q from the wells to the tank. This dz - Q pair can be used to find a particular

efficient pump configuration in the lookup table, which can be directly applied to the pumps in the optimization model.

3.4.2. Optimization Procedure

The same general-purpose MATLAB GA function was used to calculate an optimal and efficient pump schedule for 24-hour demand periods from the first day of the week-long demand periods used in the calibration procedure. The optimization of weekly operating schedules, while technically possible, was found to be computationally expensive and impractical. Actual use of such an optimization procedure would likely follow a daily schedule.

The pipe parameters and dz value from the calibration stage were applied permanently to the model, and the optimization GA was allowed to adjust the pump operating schedule in order to minimize total energy used. Early on, before the pump enumeration procedure was developed, it was necessary to use four decision variables per time step to prescribe a pump configuration. If, for example, the pump configuration is to be optimized every six hours, this initial formulation would comprise a total of 16 decision variables – four for each time step. Using this formulation, the GA was not able to converge to a meaningful solution in a reasonable amount of time. Convergence could theoretically have been achieved, but successful execution of a 24-hour optimization on an average desktop computer would take many hours, if not days.

However, through use of the pre-determined *opPts* lookup table described in the previous section, the optimization routine only requires one variable per time step:

pumped flow rate from the wells to the tank. For an optimization time step of six hours, use of the *opPts* lookup table reduces the dimensionality of the optimization from 16 decision variables to only four. An example set of these four decision variables is reproduced in Table 3-4 below.

Table 3-4. Example set of 4 decision variables for model optimization.

<u>Q1 (0-6 hrs),</u> <u>gpm</u>	<u>Q2 (6-12 hrs),</u> <u>gpm</u>	<u>Q3 (12-18 hrs),</u> <u>gpm</u>	<u>Q4 (18-24 hrs),</u> <u>gpm</u>
8,000	11,000	10,000	9,000

The optimization procedure used the same general-purpose MATLAB GA package as the calibration procedure and called the same core EPANET simulation function. For optimization purposes, the EPANET simulation function was modified to dynamically assign efficient pump configurations based on the set of decision variables Q passed to it by the GA. This set of decision variables is simply a time-series of flow in gpm that the operator wishes to pump from the wells to the tank. Static input variables that are not changed by the GA include the tank level at the beginning of the simulation and a time series of system demand applied to the demand node. With these static input variables, plus the set of decision variables Q , the EPANET simulation function is able to execute and return the total sum of energy expended by the 14 pumps in kilowatt-hours. This energy amount is the fitness value to be minimized by the GA.

The penalties for hydraulic instability that were used in the calibration procedure were also used in the optimization procedure. Additional constraints were added to the optimization model which penalized the objective function if the volume in storage at the end of the simulation was below a desired level or if water in the storage tank dipped below 150% of the minimum fire-flow volume. These penalties constrain the GA from choosing solutions which simply do not run the pumps at all, allowing whatever volume is in the tank to satisfy demand for the simulation period.

For optimization, 60% of every generation was generated by the crossover operator. The remaining 40% were generated by the mutation operator, which changed individual variables within members of the population to a new, random value with a probability of 30%. The fitness of each individual in the new generation is then evaluated, and the process repeats. A population size of 100 and a generation size of 30 was used, and the optimization procedure was executed in three consecutive trials before choosing an optimal solution.

3.5. Methodology Summary

The flowchart in Figure 3-24 below summarizes the methodology discussed in Section 3.

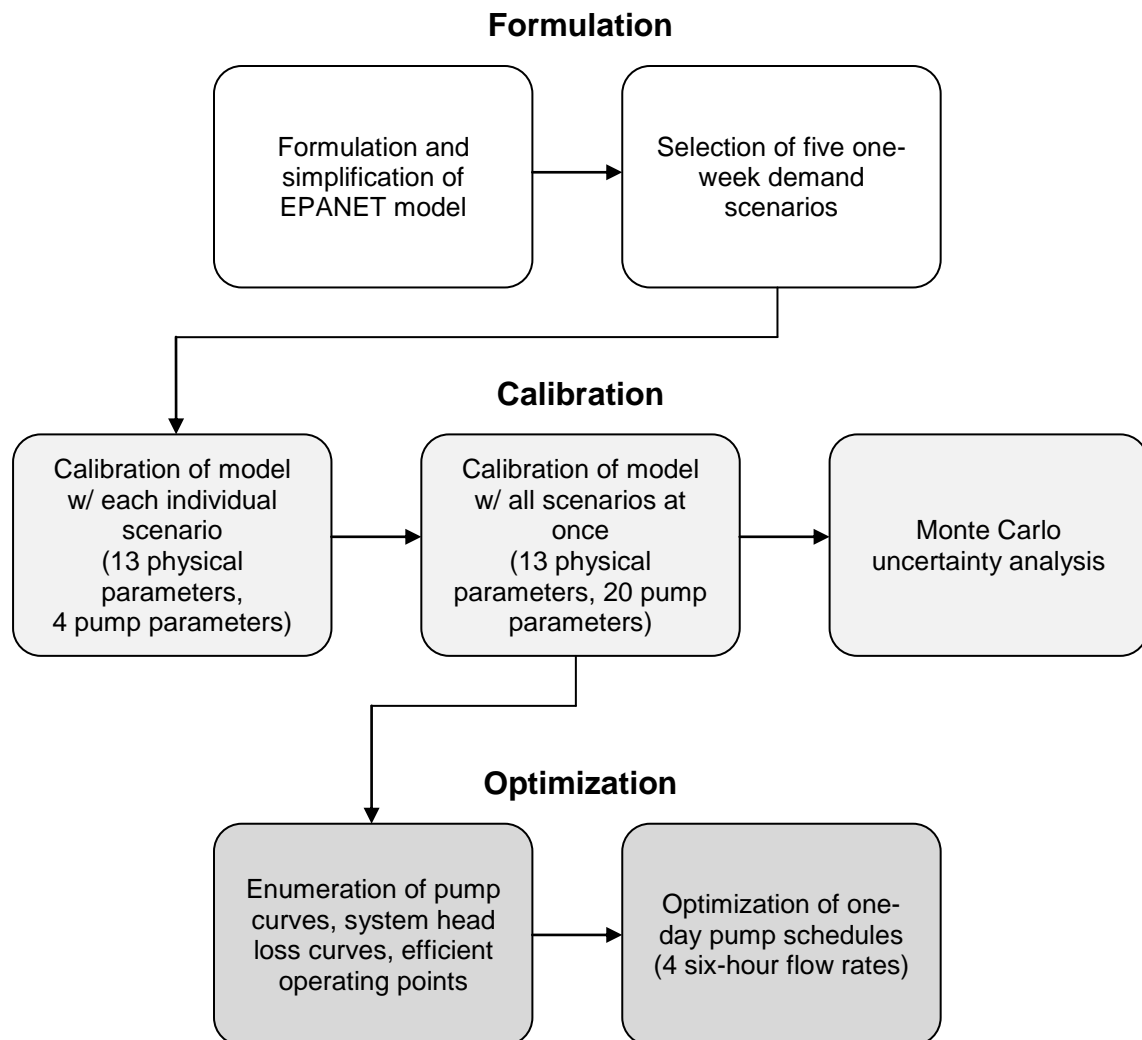


Figure 3-24. Methodology summary flowchart

4. RESULTS AND ANALYSIS

The methodology described in Section 3 yielded mixed results due to several limiting factors stemming primarily from a lack of necessary input data, which led to many simplifying assumptions made both in formulation of the EPANET model and in the number and makeup of decision variables used with the GA. These simplifying assumptions and the simultaneous calibration of all five scenarios led to unstable and inaccurate calibrations. As a result of the mixed quality of the calibrated model, optimization of pumping schedules for the first 24 hours of these scenarios was successful for only two scenarios.

4.1. Calibration Results

The final, calibrated set of the EPANET model parameters and the five sets of pump parameters for each scenario are presented in Table 4-1 below. Also presented below in Figure 4-1 through Figure 4-5 are five plots, one for each of the five week-long demand scenarios. Each plot shows the actual tank volume derived from observed SCADA information along with the tank volume calculated by the calibrated EPANET model.

Table 4-1. Calibrated set of EPANET model parameters.

<i>Pipe Properties (ClearWellIn: WP to LHP)</i>		
<u>Diameter, in</u>	<u>Length, ft</u>	<u>H-W Roughness</u>
26.0	3,973	102.2
<i>Pipe Properties (Transmission: LHP to HSP)</i>		
56.3	16,209	128.0
<i>Pipe Properties (Distribution: HSP to Storage Tank)</i>		
57.5	52,081	137.2
<i>Pipe Properties (Distribution2: Storage Tank to Demand Node)</i>		
38.0	34,952	103.8

<i>Tank Elev above Wells, ft</i>
639.1

<i>Pump Parameters (2009peak2)</i>			
<u>Initial # of WPs On</u>	<u>Initial # of LHPs On + Speed of VFD LHP</u>	<u>Initial # of Constant-Speed HSPs On</u>	<u>Initial speed of VFD HSPs</u>
6	3.10	1	0.28
<i>Pump Parameters (2009peak)</i>			
6	2.10	1	0.22
<i>Pump Parameters (2009rise)</i>			
3	1	1	0
<i>Pump Parameters (2008fall)</i>			
3	2.24	1	0.64
<i>Pump Parameters (2007flat)</i>			
3	0.60	1	0.75

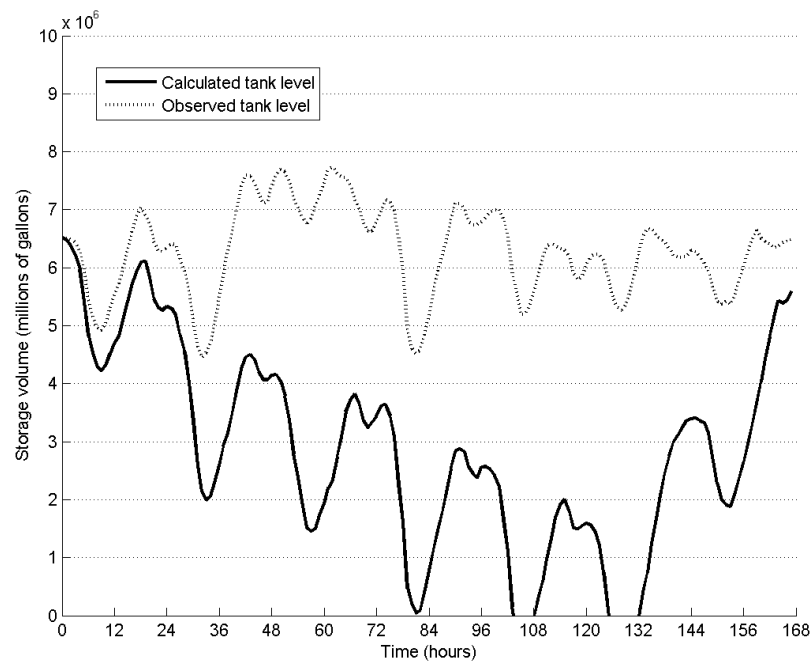


Figure 4-1. Observed vs. calculated tank volumes for Scenario 1 (2009peak2)

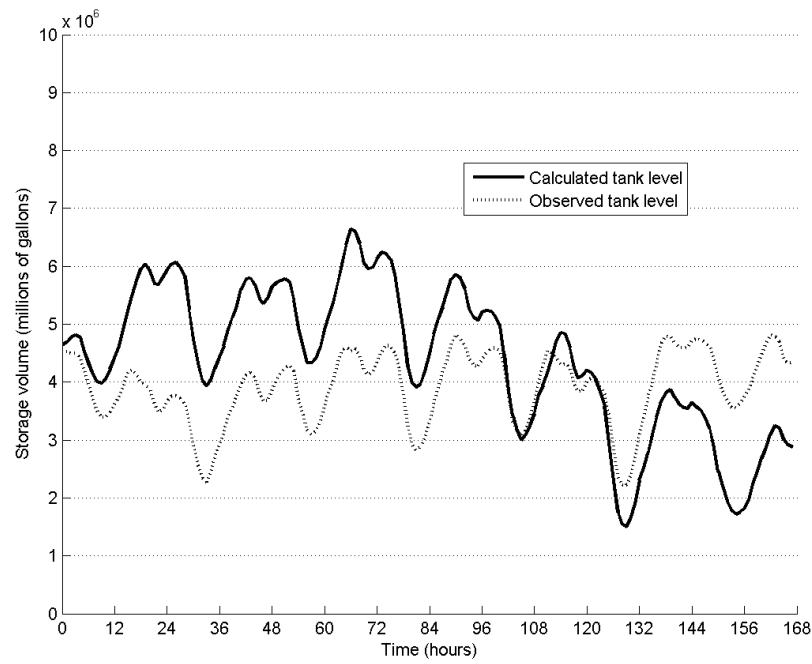


Figure 4-2. Observed vs. calculated tank volumes for Scenario 2 (2009peak)

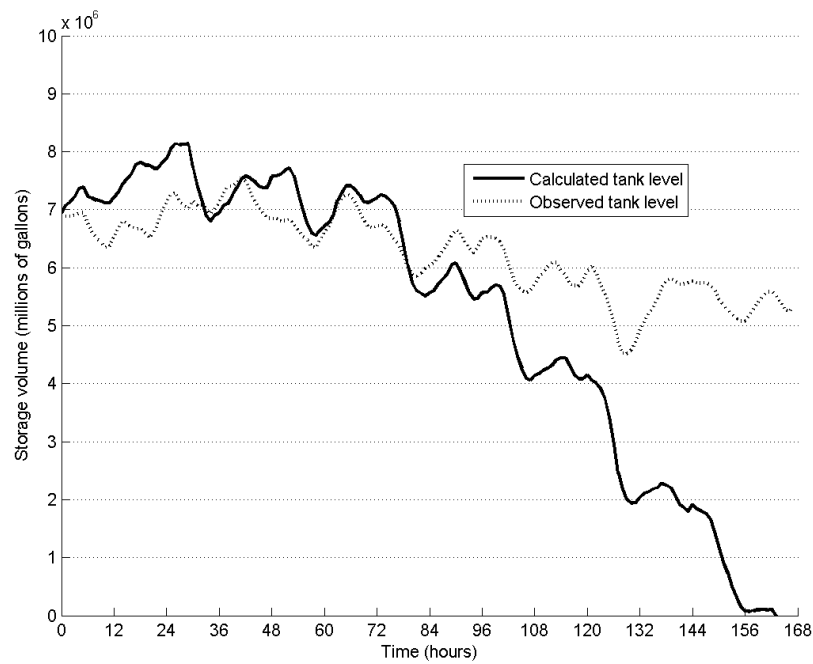


Figure 4-3. Observed vs. calculated tank volumes for Scenario 3 (2009rise)

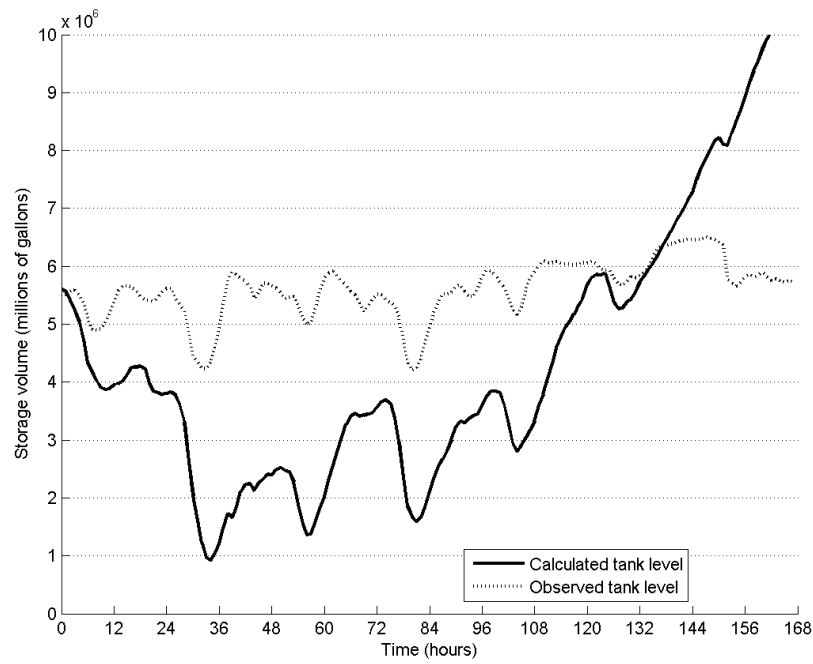


Figure 4-4. Observed vs. calculated tank volumes for Scenario 4 (2008fall)

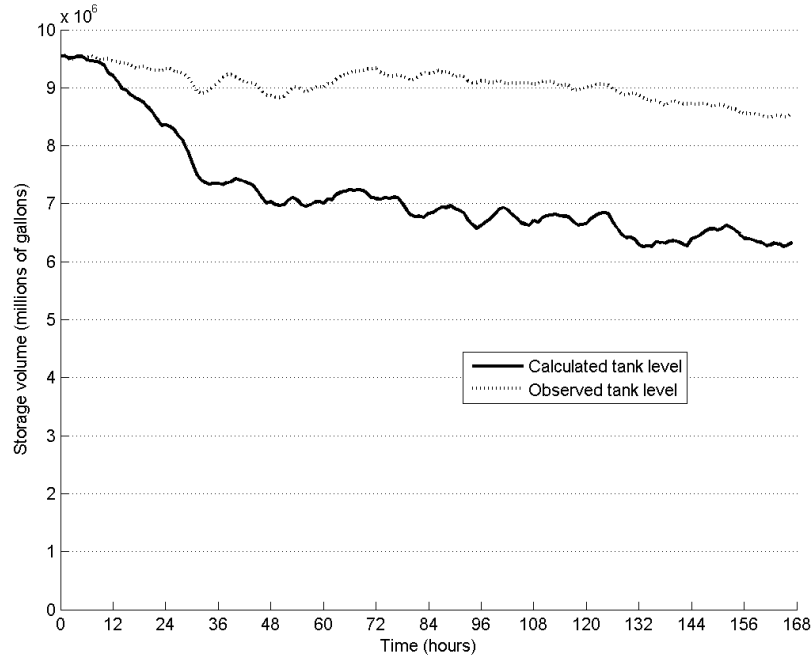


Figure 4-5. Observed vs. calculated tank volumes for Scenario 5 (2007flat)

At first glance, the calibrated parameter values appear to be physically reasonable, with the notable exception of the dz value of 639.1 ft, which is a great deal higher than the city's elevation above sea level. This distance between the WSEL of the aquifer and the minimum WSEL in the tank seems excessively high. This calibrated dz value corresponds roughly to an elevated storage tank that sits about 400 ft above ground. This corresponds to a pressure in the WDS of about 170 psi, which is much too high. Repeated execution of the calibration procedure and attempts to lower the upper bound for dz in the GA options were unsuccessful in generating better solutions, so this parameter was used despite its physical absurdity. Furthermore, two of the calibrated diameter values (56.3 in and 57.5 in) are much larger than the other two. These large diameters, and possibly the large roughness values, serve to decrease head loss in the

system, balancing out some of the problems caused by a large dz between the well and tank WSELs.

None of the figures above are very representative of a well-calibrated model, though some scenarios perform better than others. The calibrated tank level in Scenario 3 (2009rise) performs reasonably well in the first 72 hours before diverging from the observed tank level, while the levels in Scenario 1 (2009peak2) and Scenario 4 (2008fall) diverge quickly and appear wildly unstable. Scenario 2 (2009peak) appears to be one of the more stable scenarios; while it is not accurate, it remains within ± 2 million gallons of the observed level throughout all seven days of the simulation. Finally, Scenario 5 (2007flat) is also relatively stable; though its calculated level diverges from the observed level within the first 24 hours of simulation, its rate of incremental error decreases to virtually zero after that point.

Because decision variables affecting all five scenarios were calibrated simultaneously, the GA was predisposed to discard solutions which performed well for some scenarios but not for others. Recall that the five scenarios were chosen to represent a wide range of system behavior; this both increases the size of the overall decision space and decreases the feasible region of the physical parameter decision space, making successful calibration more difficult. Calibrating the parameters one scenario at a time would likely yield better results for each scenario, but these calibrated parameters could only be used for predicting WDS behavior under similar demand scenarios or weather conditions, and accuracy would still not be guaranteed. It may be the case that one or more of the dry summer or wet winter periods excludes an area of

the decision space that would have held a successful solution had these edge cases not been considered.

Finally, the simplifying assumptions made in the early stages of the EPANET model development and in the formulation of the decision variables may have made successful calibration impossible. These assumptions included a minimum volume used when developing the detrended tank volume time series, an assumed combined cylindrical tank diameter of 150 feet, that both VFD-equipped HSPs would run at the same speed, and that the pump configuration remained static across the entire week for each calibrated scenario.

Note that no other independent observed data was available for comparison. Even if the GA were able to calibrate the WDS parameters such that tank level could be accurately predicted, pump energy expenditure calculated using such parameters may not necessarily match observed energy expenditure, if such data were available.

4.2. Uncertainty Analysis

As Scenarios 3 and 5 (2009rise and 2007flat, respectively) were the only successfully optimized scenarios in the following Optimization Results section, they are the only scenarios presented as part of the uncertainty analysis. Plots showing the results of the Monte Carlo uncertainty analysis are presented in Figure 4-6 through Figure 4-13 on the next four pages. The observed tank level for each scenario is plotted in bold, while the mean calculated tank level is plotted as a solid line surrounded by the minimum, 25th percentile, 75th percentile, and maximum tank levels calculated during the analysis.

Scenario 5 (2007flat) appears to be very sensitive to small fluctuations in the model parameters. Figure 4-10, for example, demonstrates that the observed tank level remains below the 75th percentile in a 10% uncertainty analysis of the calibrated parameters. Recall that the Monte Carlo analysis in Figure 4-10 was executed 200 consecutive times, with each parameter varying along a normal distribution in which two standard deviations from the calibrated value lie within 10% of the original calibrated value.

Scenario 3 (2009rise), on the other hand, does not appear to be very sensitive to small fluctuations (10%) in the model parameters: the 25th and 75th percentiles, as well as the maximum, are all situated tightly around the mean. As uncertainty increases, the distribution of the observed tank levels spreads out and the observed tank level finally falls within the maximum calculated tank level at 30% uncertainty. This demonstrates that the parameters are poorly calibrated for Scenario 3 across the entire week.

However, if one considers only the first 24 hours of the simulated period, the parameters appear to be calibrated reasonably well for Scenario 3 within a 10-20% level of certainty for each individual parameter.

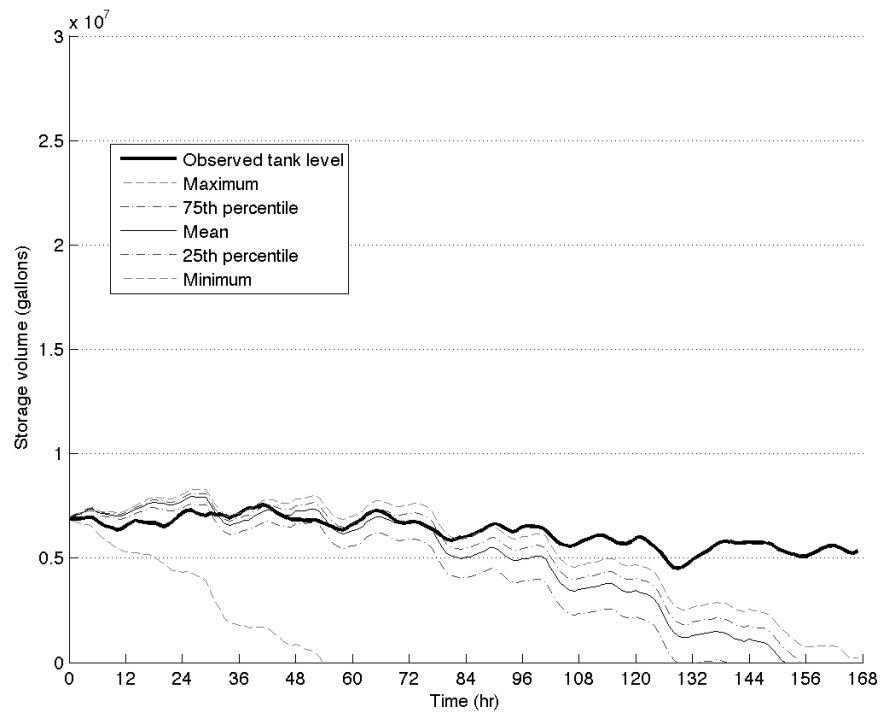


Figure 4-6. Monte Carlo, Scenario 3 (2009rise), 10% uncertainty

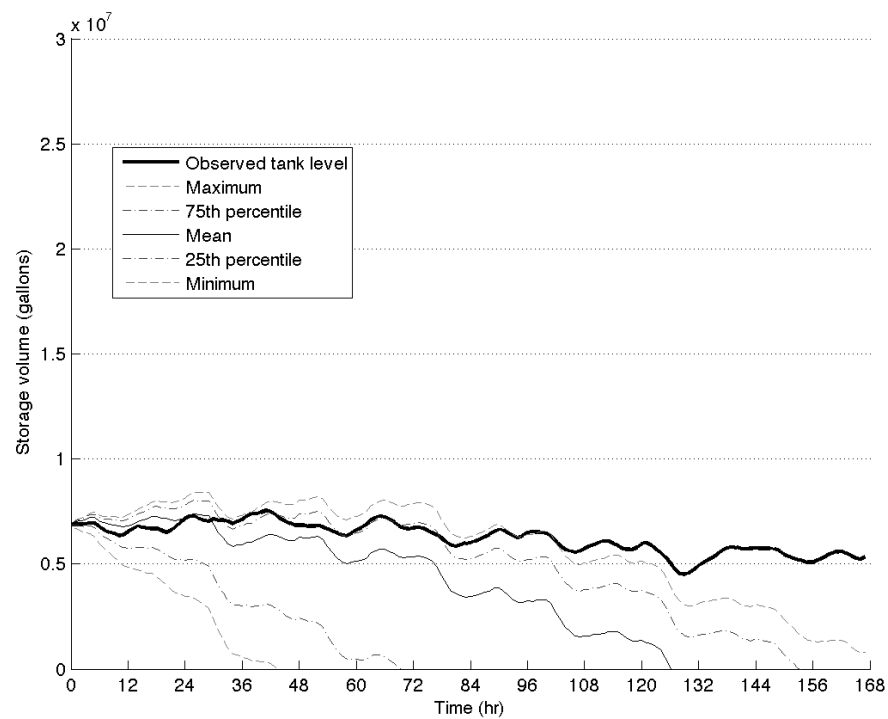


Figure 4-7. Monte Carlo, Scenario 3 (2009rise), 20% uncertainty

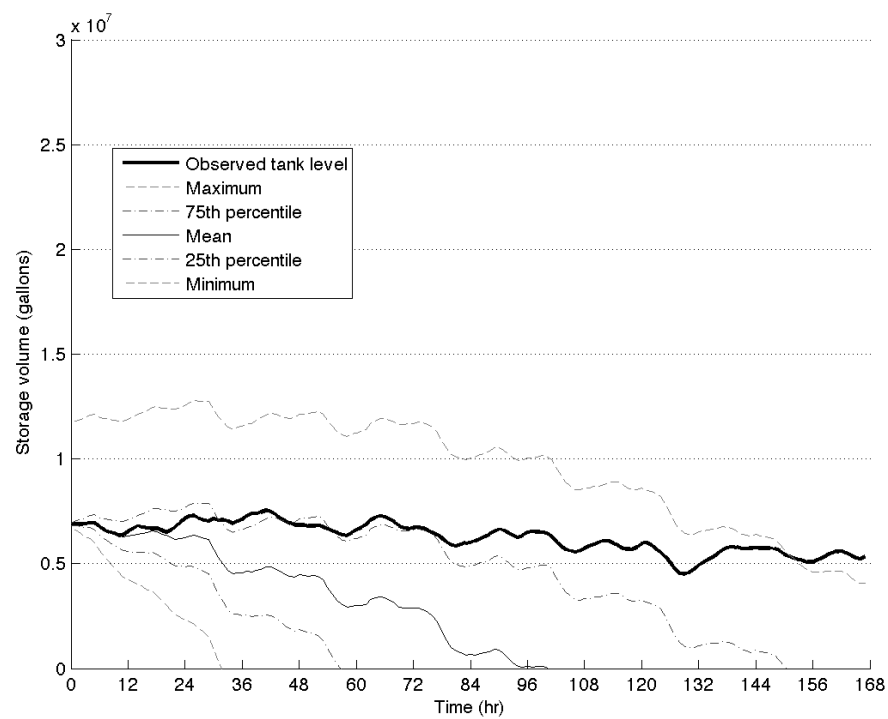


Figure 4-8. Monte Carlo, Scenario 3 (2009rise), 30% uncertainty

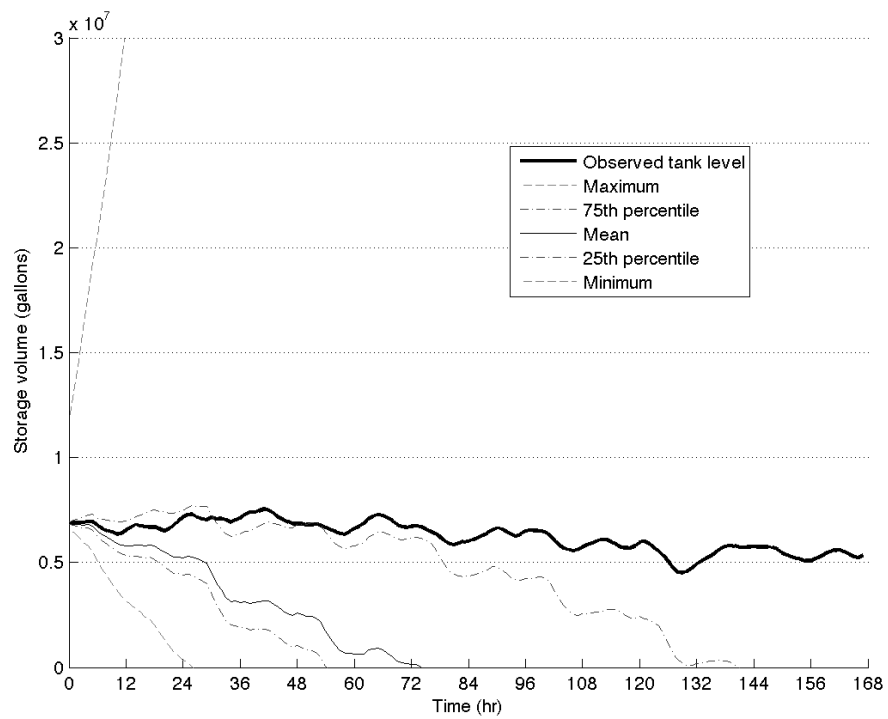


Figure 4-9. Monte Carlo, Scenario 3 (2009rise), 40% uncertainty

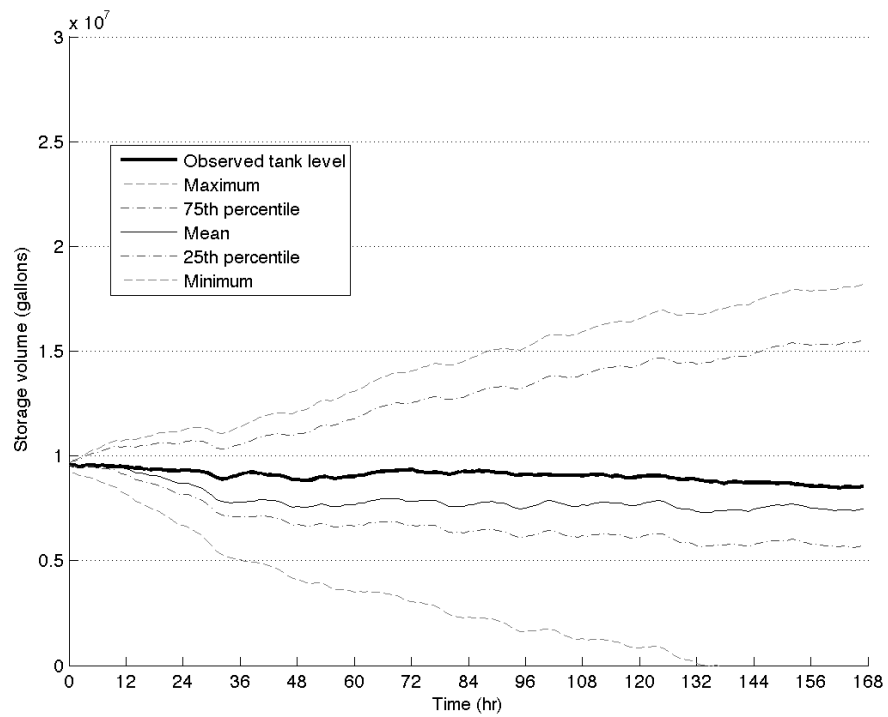


Figure 4-10. Monte Carlo, Scenario 5 (2007flat), 10% uncertainty

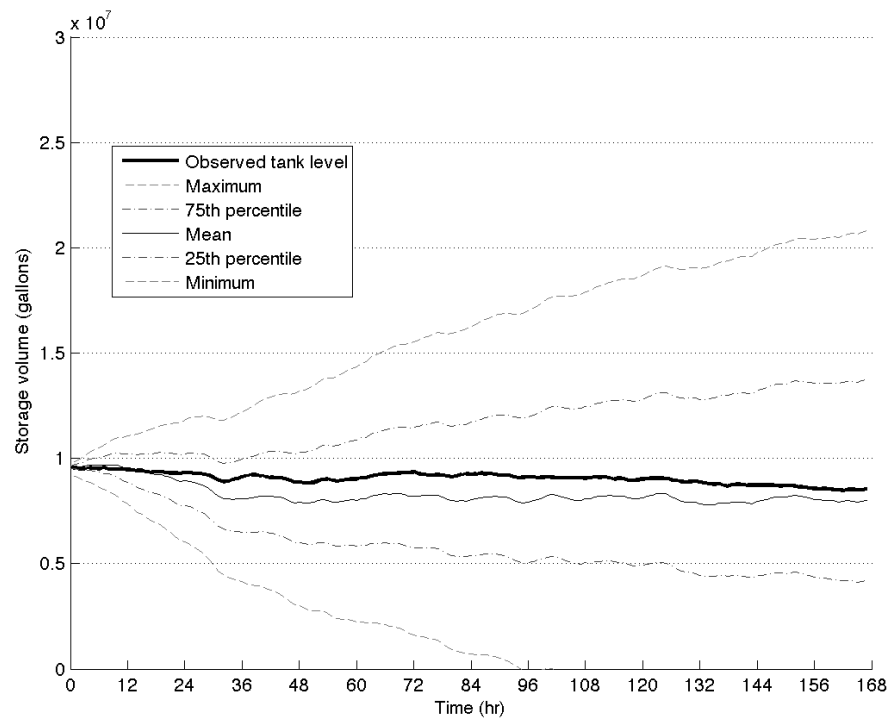


Figure 4-11. Monte Carlo, Scenario 5 (2007flat), 20% uncertainty

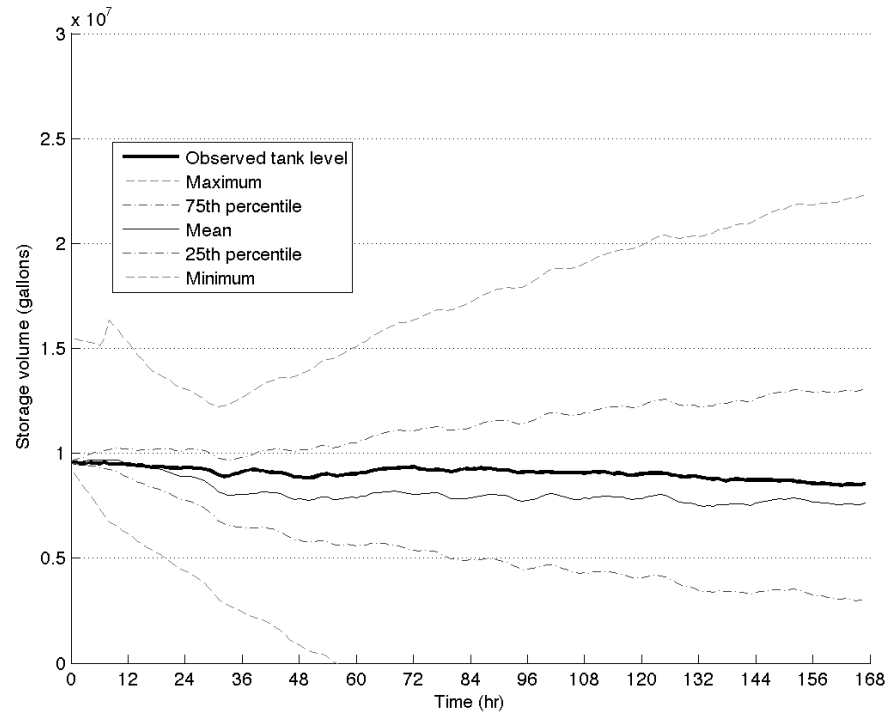


Figure 4-12. Monte Carlo, Scenario 5 (2007flat), 30% uncertainty

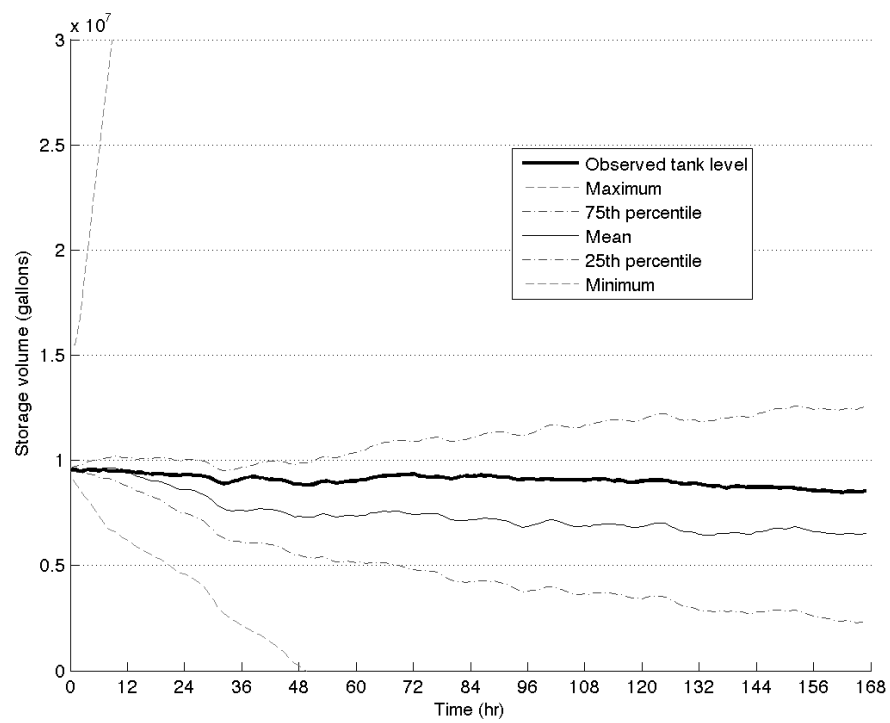


Figure 4-13. Monte Carlo, Scenario 5 (2007flat), 40% uncertainty

4.3. Optimization Results

Optimization was ultimately unsuccessful for Scenarios 1, 2, and 4 – that is, the summertime scenarios 2009peak2 and 2009peak and the falling-demand transition scenario 2008fall, respectively. In these cases, the GA was unable to avoid the penalties functions assessed by the simulation function for head increases across pumps exceeding the cutoff head or for flows exceeding maximum capacity. Examination of the calibrated parameters for these scenarios revealed that similar penalties had been assessed during calibration. Refer to the calibrated tank level plots for the week-long simulation in Figure 4-1, Figure 4-2, and Figure 4-4 on pages 59 to 60. Note the wildly varying calibrated tank level, especially in Figure 4-1 and Figure 4-4. This large deviation from expected values led to many EPANET hydraulic errors. The fact that EPANET continues to report errors after optimization of the pump parameters – even in a short 24-hour period – suggests that the physical parameters are unrealistic. Because of the errors in these scenarios, the optimization output regarding energy and efficiency is inaccurate, rendering these scenarios ineffective as candidates for optimization.

The optimized model output for the first day of Scenarios 3 and 5 (2009rise and 2007flat) is presented below in Table 4-2 through Table 4-5. The column Q (gpm) contains the four decision variables used in the optimization GA. For the optimized scenarios, the *Pump Parameters* column contains the parameters pre-determined to be efficient for the particular combination of dz and Q at the beginning of the optimization time step. For the calibrated scenarios, the *Pump Parameters* column simply contains the calibrated pump parameters.

Table 4-2. Optimized parameters and energy statistics, Scenario 3 (2009rise).

t (hr)	Q (gpm)	Pump Parameters				Energy Used (kWh)	Efficiency (%)
0-6	9,469	3	1	1	0	9,224	84.6
6-12	4,661	2	0.6	0	0.8	6,163	79.2
12-18	9,401	3	1	1	0	8,617	84.3
18-24	8,349	3	0.8	1	0	8,079	85.6
						32,083	84.0
							Overall

Table 4-3. Calibrated parameters and energy statistics, Scenario 3 (2009rise).

t (hr)	Q (gpm)	Pump Parameters				Energy Used (kWh)	Efficiency (%)
0-6	9,504	3	1	1	0	9,224	84.6
6-12	9,496	3	1	1	0	9,224	84.6
12-18	9,485	3	1	1	0	9,226	84.6
18-24	9,457	3	1	1	0	9,234	84.7
						36,908	84.6
							Overall

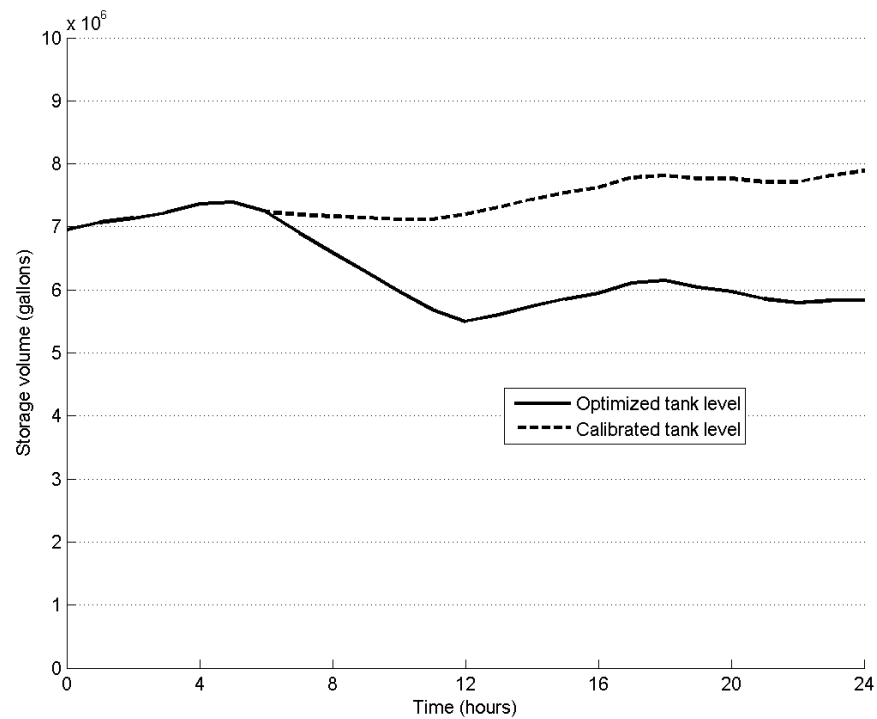


Figure 4-14. Calibrated and optimized tank levels, Scenario 3 (2009rise)

Table 4-4. Optimized parameters and energy statistics, Scenario 5 (2007flat).

<u>t (hr)</u>	<u>Q (gpm)</u>	<u>Pump Parameters</u>				<u>Energy Used (kWh)</u>	<u>Efficiency (%)</u>	
0-6	6,495	2	0.8	1	0	6,498	83.7	
6-12	5,275	3	0.6	0	0.8	6,617	74.6	
12-18	6,504	2	0.8	1	0	6,521	83.7	
18-24	6,354	2	0.8	1	0	6,498	83.7	
						26,134	81.8	Overall

Table 4-5. Calibrated parameters and energy statistics, Scenario 5 (2007flat).

<u>t (hr)</u>	<u>Q (gpm)</u>	<u>Pump Parameters</u>				<u>Energy Used (kWh)</u>	<u>Efficiency (%)</u>	
0-6	6,826	3	0.6	1	0	6,895	82.8	
6-12	6,766	3	0.6	1	0	6,902	82.6	
12-18	6,743	3	0.6	1	0	6,906	82.6	
18-24	6,724	3	0.6	1	0	6,906	82.5	
						27,609	82.6	Overall

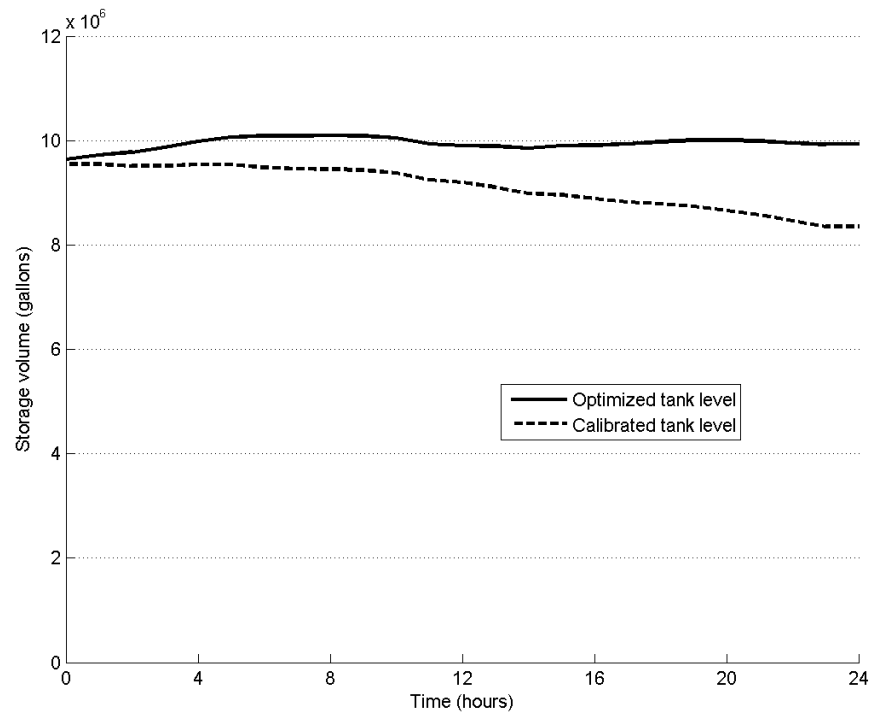


Figure 4-15. Calibrated and optimized tank levels, Scenario 5 (2007flat)

See Figure 4-14 and Figure 4-15 above for plots of the calibrated and optimized tank levels for each scenario. Note that the only differences between the input data to the optimized run and to the calibrated run are the pump parameters used.

Compare the overall hydraulic efficiencies between the calibrated and optimized scenarios. Although the overall efficiency of the pumps in the calibrated scenarios (84.6% and 82.6%) are lower than the overall efficiencies of the pumps in the optimized scenarios (84.8% and 81.8%), less energy is used in the optimized scenarios than in the calibrated ones. This may seem counterintuitive, but the end goal is the minimization of energy usage, not necessarily the maximization of efficiency. The two do not necessarily coincide, as many pumps operating at a higher efficiency will use more energy than fewer pumps operating at a slightly lower efficiency.

The optimization of the pump parameters is a success for these two scenarios. In Day 1 of Scenario 3 (2009rise), 4,825 kWh of energy is saved – a savings of 13.1%. In Day 1 of Scenario 5 (2007flat), 1,475 kWh of energy is saved – a savings of 5.3%. At a typical utility rate of \$0.10/kWh, this corresponds to an energy savings of roughly \$483/day and \$148/day compared to the calibrated base case.

4.4. Caveats

As mentioned previously, major caveats to the work presented here are the simplifying assumptions, mostly stemming from limited input data. Aggregate flow rates into and out of storage and aggregate flow rates out of the HSP station were the only long-term SCADA-recorded information available. Although flow rates into and out of storage were adjusted to remove an upward trending effect, there is no way to check this assumption. A similar trend may have affected the flow rates at the HSP station, but there is no way to address this without more data. The lack of a time series of absolute tank volume necessitated an assumption about minimum tank volume during the detrending process. Also, the WDS considered in this work actually contains more than one elevated storage tank. The tanks were modeled in aggregate because of the input data that was provided, but modeling the tanks separately, though it would increase the number of decision variables in the calibration stage, may lead to more accurate results.

The lack of flow data recorded for individual pumps led to an assumption that the pump statuses in the calibrated models would remain static for the duration of the week-long simulation. An early calibration approach added simple logic to the simulator that would periodically check if pumps should be turned on or off to maintain a decent level of efficiency. However, this approach essentially allowed the model to optimize itself during the calibration stage, leaving little room to examine the effects of the optimization model. Therefore, the pump configuration was assumed to remain unchanged for the duration of the simulation.

Other caveats stem from the choice and formulation of decision variables. The diameter of the aggregate elevated tank, for example, should have been a calibrated variable because it controls the relation of volume in storage to hydraulic head at the tank. As of version 2.00.12a, however, the EPANET Toolkit's function for changing tank diameter is not functional. A diameter of 150 ft was chosen as an educated guess and tested by trial and error during multiple calibration runs, but adding it as a decision variable in the calibration stage may result in another value leading to a more well-calibrated model.

The calibration procedure could also be made more efficient through smarter decision variable formulation or possibly the use of an optimization algorithm other than GA. The calibration GA implemented in this work does not differentiate between physical pipe parameters, the dz value, or the pump parameters, even though each of these types of parameters may have varying degrees of effect on the quality of calibration. The crossover operator could be modified to consider a group of three pipe parameters as one chromosome rather than three, for example, or to consider the pump parameters as a single chromosome. Also, because a GA generates a diverse set of solutions at once and does not evaluate gradients in search of optimal solutions, it is by nature more of an exploratory algorithm than an exploitative one. That is, a GA is good for searching a very large decision space for areas containing feasible solutions. Once a feasible region is found, a GA is not particularly well-suited to drilling down and quickly finding the local optimum of that feasible region. Pairing the GA with a simulated

annealing algorithm or even a much simpler algorithm that follows the steepest gradients might improve the results considerably.

In the pump enumeration stage, the VFD-equipped pumps were assumed to run at coarsely discrete speeds ranging from 20% to 100% in 20%-increments. This formulation was chosen because of memory and computation time constraints in an effort to limit the number of characteristic curves that needed to be calculated and intersected with the system head loss curves. As a result of this, the optimization algorithm was able to assign pump speeds of 60%, 80%, and 100%, for example, but was not able to assign a pump speed of 90%. For the same reason, both of the VFD-equipped HSPs were assumed to run together at the same speed. These caveats in the pump parameter formulation limit the efficiency of the pump combinations applied during the optimization stage, and overall efficiency in the optimization stage may be improved by increasing the number of discrete VFD speeds used.

Finally, note that the reported energy savings of \$483/day and \$148/day depend heavily on the assumption that the pump configuration did not change during the weekly scenario. A more accurate estimation of energy savings could be made by calibrating the original pump configuration of each model to the first 24 hours of the discussed scenarios, rather than the full week. Accuracy of energy expenditure calculations would be most improved if the original pump configuration and/or the original energy expenditure of each pump station had been recorded along with the rest of the available SCADA information. Future studies should ensure that such data is recorded and included in the calibration process.

5. CONCLUSIONS

The minimization of energy use in a water distribution system is an economic and environmental imperative for water utility providers. Money saved in energy usage can be put toward infrastructure improvements or passed on to the consumer in the form of lower usage rates, while lowering the carbon footprint of a water distribution system has long-term positive environmental effects. Many possible alternatives to minimizing energy use can be explored through hydraulic modeling. Modeling a given water distribution system and calibrating its parameters necessitates certain simplifications to the system which compromise the ability of the model to accurately predict the actual system's behavior, but effective formulation of parameters and optimization algorithms can reduce these ill effects. The balancing act between modeling the system accurately and simplifying its parameters for ease of development and use is complex and successful calibration techniques will differ from system to system.

In this work, a hydraulic model of a water distribution system was developed using EPANET software and repeatedly simulated to calibrate its physical and operational parameters such that calculated tank volume would be as close as possible to observed tank volume for five scenarios. This calibrated model was used to optimize pump scheduling for two of those five scenarios in order to minimize the daily expenditure of pump energy for 24-hour periods.

This methodology does not provide an operator with a straightforward calculator that finds the globally cheapest pump operation schedule for any WDS model. However, an engineer or pump operator with a well-calibrated hydraulic model can

apply this approach to his own distribution system, optimize the pump operating schedule for an anticipated demand period, and compare its results to a planned pump operation schedule to make an informed decision regarding day-to-day pump operations.

The methodology and results of this thesis could be most improved by gathering additional data from the water utility to include in the calibration process. The aggregate flow rate data used in this work is a good start for creating a simplified network model, but this research has shown that more detailed information is needed to create a model that is well-calibrated for calculating energy usage and tank levels. To assist in formulating an improved network model, the water utility may consider using its SCADA system to record and archive time series of the following operational data:

- Volume of water and WSEL in each storage tank
- Status and VFD speed of each pump
- Flow rates through each pump
- Power usage at each pump station
- WSEL in each well

This data should be recorded and archived by the utility, ideally at one-hour or shorter time intervals. Longer time intervals may be satisfactory for some variables, especially the power usage at each pump station. Accounting for this data in the calibration process would allow the modeler to avoid many of the simplifying assumptions made early in the formulation and calibration stages, especially the assumptions regarding long pump schedules and regarding minimum tank volume

during the “detrending” process. Finally, the archiving of power usage at each pump station would provide another metric for evaluating the accuracy of the calibrated model. Because this work demonstrates potential energy savings of perhaps hundreds of dollars per day, the importance of archiving as much SCADA information as possible to improve the model calibration and optimization process cannot be overstated.

The overall methodology presented here could also be improved by:

- Exploring alternative optimization methods when calibrating the physical and operational parameters and when optimizing the calibrated model to minimize energy usage. Methods not explored here that may be more efficient include simulated annealing, ant-colony optimization, or a hybrid GA-“hillclimber” method approach, similar to the one used by van Zyl et al. (2004).
- Developing a more detailed EPANET model to account for variation in volume and WSELs in separate tanks and separate wells. This step would need to be conducted in conjunction with more comprehensive SCADA information provided by the water utility.
- Adding additional stages of calibration targeting different types of input parameters individually, *or* adding a calibration stage that scales different groups of input parameters together. For example, the full, 33-parameter calibrated model could be re-calibrated by scaling all pipe roughness parameters together using one decision variable, scaling all pipe diameter parameters using another decision variable, etc. These steps may be a

good way of quickly refining the ballpark parameters from earlier calibration stages.

- Enumerating a greater number of pump combination curves at a higher resolution. Instead of including all VFD speeds from 0% to 100% at 20% intervals, the modeler could include VFD speeds from 50% to 100% at 10% intervals or smaller. This would result in a higher-precision $dz-Q$ lookup table for the optimization procedure, which may result in a more efficient set of operating points.

Future research in this area could examine the more complicated objective of minimizing actual pumping cost rather than simply the pumping energy used. Some energy providers charge different amounts per kilowatt-hour at different times of day; for example, energy costs may be lower at night than they are during peak demand times in the morning or afternoon. These price factors may range between roughly 40% and 200% of the average daily rate (Li and Flynn 2004) which, if applied to this work, would correspond to a minimum rate of \$0.04/kWh around 4:00 AM and a maximum rate of \$0.20/kWh at 12:00 PM and/or 6:00 PM. These diurnal pricing schemes could be accounted for in the optimization stage and would likely lead to pump schedules biased toward night-time pumping, depending on the pricing scheme employed by the WDS' utility provider. Future research could also focus on more long-term possibilities for system optimization, such as installing additional ground-level storage tanks between the well pumps and low-head transmission pumps, adding or upgrading pumps to the system, or adding additional storage tanks to the distribution system. Adding additional

tanks between the pump stations could allow the stations to be operated independently of one another, avoiding situations in which one pump station may have to operate at a lower efficiency than it is capable of operating at independently. Several alternatives could be developed, and their optimized results could be compared with the current system. A cost-benefit analysis would need to be performed to determine the relative value of each long-term design alternative.

REFERENCES

- Brion, L. M., and Mays, L. W. (1991). "Methodology for optimal operation of pumping stations in water distribution systems." *Journal of Hydraulic Engineering*, 117(11), 1551-1569.
- Farmani, R., Walters, A., and Savic, D. (2005). "Trade-off between total cost and reliability for Anytown water distribution network." *Journal of Water Resources Planning and Management*, 131(3), 161-171.
- Hanselman, D., and Littlefield, B. (2001). *Mastering MATLAB 6*, Prentice Hall, Upper Saddle River, New Jersey.
- Jeong, H. S., and Abraham, D. M. (2009). "Water Rationing Model for Consequence Minimization of Water Infrastructure Destruction." *Journal of Water Resources Planning and Management*, 135(2), 80-89.
- Li, Y., and Flynn, P. C. (2004). "Deregulated power prices: Comparison of diurnal patterns." *Energy Policy*, 32(5), 657-672.
- Lopez-Ibanez, M., Prasad, T. D., and Paechter, B. (2008). "Ant colony optimization for optimal control of pumps in water distribution networks." *Journal of Water Resources Planning and Management*, 134(4), 337-346.
- Ormsbee, L. E., and Lansey, K. E. (1994). "Optimal control of water-supply pumping systems." *Journal of Water Resources Planning and Management*, 120(2), 237-252.
- Rossman, L. A. (2000). *EPANET 2 User's Manual*. Water Supply and Water Resources Division, ed., United States Environmental Protection Agency, Cincinnati, OH.

- Sakarya, A. B. A., and Mays, L. W. (2000). "Optimal operation of water distribution pumps considering water quality." *Journal of Water Resources Planning and Management*, 126(4), 210-220.
- Simpson, A. R., Dandy, G. C., and Murphy, L. J. (1994). "Genetic algorithms compared to other techniques for pipe optimization." *Journal of Water Resources Planning and Management*, 120(4), 423-443.
- Ulanicki, B., Kahler, J., and See, H. (2007). "Dynamic optimization approach for solving an optimal scheduling problem in water distribution systems." *Journal of Water Resources Planning and Management*, 133(1), 23-32.
- van Zyl, J. E., Savic, D. A., and Walters, G. A. (2004). "Operational optimization of water distribution systems using a hybrid genetic algorithm." *Journal of Water Resources Planning and Management*, 130(2), 160-170.

APPENDIX A EPANET MODEL

```

[JUNCTIONS]
;ID      Elev      Demand      Pattern
LH1      0          0.000000    Day1          ;elev328
LH2      0          0.000000    Day1          ;
LH3      0          0.000000    Day1          ;elev328
HS1      0          0.000000    Day1          ;elev305
HS2      0          0.000000    Day1          ;
HS3      0          0.000000    Day1          ;elev305
DEM      0          9000        1             ;
1        0          0             ;
2        0          0             ;

[RESERVOIRS]
;ID      Head      Pattern
Well1    0          ;
Well2    0          ;
Well3    0          ;
Well4    0          ;
Well5    0          ;
Well6    0          ;

[TANKS]
;ID      Elevation  InitLevel  MinLevel  MaxLevel  Diameter  MinVol  VolCurve
ElevTank 0          504.8      0         999.0000  150       0       ;elev295

[PIPES]
;ID      Node1      Node2      Length  Diameter  Roughness  MinorLoss
Status
ClearWellIn  LH1      1          1283.96  49.93     191.18     0.0000
CV          ;
ClearWellOut  1          LH2      1283.96  49.93     191.18     0.0000
CV          ;
Transmission LH3      HS1      26143.22  39.80     79.09      0.0000
CV          ;

```


GroundTankIn	HS1	2	26143.22	39.80	79.09	0.0000
CV ;						
GroundTankOut	2	HS2	26143.22	39.80	79.09	0.0000
CV ;						
Distribution	HS3	ElevTank	16981.64	60.89	177.25	0.0000
CV ;						
Distribution2	ElevTank	DEM	1	100	100	0.0000
Open ;						

[PUMPS]

;ID	Node1	Node2	Parameters
WellPump1	Well11	LH1	HEAD Well13 ;
WellPump2	Well12	LH1	HEAD Well15 ;
WellPump3	Well13	LH1	HEAD Well13 ;
WellPump4	Well14	LH1	HEAD Well15 ;
WellPump5	Well15	LH1	HEAD Well13 ;
WellPump6	Well16	LH1	HEAD Well15 ;
LHP1	LH2	LH3	HEAD LHP ;
LHP2	LH2	LH3	HEAD LHP ;
LHP3	LH2	LH3	HEAD LHP ;
LHP4	LH2	LH3	HEAD LHP ;
HSP1	HS2	HS3	HEAD HSP1 SPEED .8 ;
HSP2	HS2	HS3	HEAD HSP2 ;
HSP3	HS2	HS3	HEAD HSP1 ;
HSP4	HS2	HS3	HEAD HSP2 ;

[DEMANDS]

;Junction	Demand	Pattern	Category
-----------	--------	---------	----------

[STATUS]

;ID	Status/Setting
WellPump3	Closed
WellPump4	Closed
WellPump5	Closed
WellPump6	Closed
LHP2	Closed
LHP3	Closed
LHP4	Closed
HSP1	.8
HSP2	Closed
HSP3	Closed
HSP4	Closed

```

[PATTERNS]
;ID          Multipliers
;
Day1         0.8387      0.7121      0.8289      0.8254      0.6906      0.9623
Day1         1.1758      0.9950      0.9774      0.9797      0.9832      0.9236
Day1         0.8043      0.7625      0.7200      0.7680      0.7734      0.7006
Day1         0.9141      1.0027      0.9245      1.0317      0.9340      0.7488
;
Day2         0.5934      0.6549      0.7986      0.9345      0.8984      0.6543
Day2         1.5805      1.4320      1.3079      1.3462      0.9793      0.7591
Day2         0.8117      0.6586      0.9023      0.6695      0.7077      0.6953
Day2         0.7916      0.9956      1.0084      1.0558      0.9335      0.8674
;
Day3         0.8594      0.9091      0.8541      0.7782      0.8689      1.1870
Day3         1.3792      1.3885      1.2406      1.2014      0.9492      0.7646
Day3         0.7740      0.8346      0.7297      0.4464      0.7021      0.7322
Day3         0.9304      1.0268      1.0868      1.0517      0.9197      0.8315
;
Day4         0.7623      0.8228      0.9870      1.0291      1.1432      1.3541
Day4         1.5604      1.5315      1.3446      1.1322      0.9823      0.9924
Day4         0.7927      0.8780      0.7025      0.7952      0.7705      0.6614
Day4         0.8823      1.2041      1.1986      1.2431      1.0478      0.9313
;
Day5         0.8025      0.9058      0.8028      0.8470      0.9701      1.2422
Day5         1.5870      1.5538      1.3647      1.2259      1.2001      0.9850
Day5         0.7618      0.8941      0.7975      0.8962      0.8544      0.7875
Day5         0.8665      0.9575      1.0652      1.2664      1.0326      0.9019
;
Day6         0.7860      1.0891      1.0558      1.0155      1.3023      1.5332
Day6         1.6863      1.7651      1.3045      1.2432      1.0322      0.9084
Day6         0.8282      0.8682      0.8773      0.8171      0.9631      0.7661
Day6         1.0162      1.0288      1.1268      1.2064      1.0569      1.0358
;
Day7         0.9442      1.0833      1.0061      1.0254      1.1104      1.3807
Day7         1.3843      1.3361      1.2674      1.3332      1.2366      1.2253
Day7         1.0547      0.9841      0.9070      0.9474      0.9494      0.9900
Day7         0.9224      1.1604      1.3137      1.3292      1.0171      1.0017
;
1            1

```

[CURVES]

;ID	X-Value	Y-Value
;PUMP: PUMP:		
HSP1	0.0000	277.0000
HSP1	4000.0000	260.0000
HSP1	6000.0000	235.0000
HSP1	6800.0000	220.0000
HSP1	8000.0000	200.0000
HSP1	8800.0000	190.0000
HSP1	11400.0000	135.0000
;PUMP: PUMP:		
HSP2	0.0000	250.0000
HSP2	2600.0000	245.0000
HSP2	3700.0000	240.0000
HSP2	4550.0000	230.0000
HSP2	5450.0000	210.0000
HSP2	6250.0000	195.0000
HSP2	6900.0000	170.0000
HSP2	7400.0000	155.0000
HSP2	8200.0000	115.0000
;PUMP: PUMP:		
HSP1eff	0.0000	0.0000
HSP1eff	4000.0000	64.3000
HSP1eff	6000.0000	82.0000
HSP1eff	6800.0000	86.0000
HSP1eff	8000.0000	89.0000
HSP1eff	8800.0000	88.0000
HSP1eff	11400.0000	81.0000
;PUMP: PUMP:		
HSP2eff	0.0000	0.0000
HSP2eff	2600.0000	62.0000
HSP2eff	3700.0000	74.0000
HSP2eff	4550.0000	80.0000
HSP2eff	5450.0000	84.0000
HSP2eff	6250.0000	84.0000
HSP2eff	6900.0000	82.0000
HSP2eff	7400.0000	78.0000
HSP2eff	8200.0000	65.0000
;PUMP: PUMP:		
LHP	0.0000	330.0000
LHP	1800.0000	310.0000
LHP	3700.0000	270.0000
LHP	5800.0000	240.0000

LHP	7400.0000	220.0000
LHP	8200.0000	200.0000
LHP	9000.0000	180.0000
LHP	10200.0000	140.0000
LHP	11200.0000	100.0000
;PUMP: PUMP:		
LHPeff	0.0000	0.0000
LHPeff	1800.0000	32.0000
LHPeff	3700.0000	57.0000
LHPeff	5800.0000	76.0000
LHPeff	7400.0000	84.0000
LHPeff	8200.0000	86.0000
LHPeff	9000.0000	88.0000
LHPeff	10200.0000	83.0000
LHPeff	11200.0000	73.0000
;PUMP: PUMP:		
Well3	0	560
Well3	2200	500
Well3	2971	414
;PUMP: EFFICIENCY:		
Well3eff	0	0
Well3eff	1000	54
Well3eff	2000	79
Well3eff	2800	85
Well3eff	3400	80
Well3eff	3900	69
;PUMP: PUMP:		
Well5	0.0000	505
Well5	1000	500
Well5	3000	400
;PUMP: EFFICIENCY:		
Well5eff	0	0
Well5eff	1000	50
Well5eff	2000	76
Well5eff	2900	85
Well5eff	3600	80
Well5eff	4000	71
[ENERGY]		
Global Efficiency	75.0000	
Global Price	0	
Demand Charge	0.0000	

Pump	WellPump1	Efficiency	Well3eff
Pump	WellPump2	Efficiency	Well5eff
Pump	WellPump3	Efficiency	Well3eff
Pump	WellPump4	Efficiency	Well5eff
Pump	WellPump5	Efficiency	Well3eff
Pump	WellPump6	Efficiency	Well5eff
Pump	LHP1	Efficiency	LHPeff
Pump	LHP2	Efficiency	LHPeff
Pump	LHP3	Efficiency	LHPeff
Pump	LHP4	Efficiency	LHPeff
Pump	HSP1	Efficiency	HSP1eff
Pump	HSP2	Efficiency	HSP2eff
Pump	HSP3	Efficiency	HSP1eff
Pump	HSP4	Efficiency	HSP2eff

[REACTIONS]

Order Bulk	1.00
Order Tank	1.00
Order Wall	1
Global Bulk	0.000000
Global Wall	0.000000
Limiting Potential	0
Roughness Correlation	0

[TIMES]

Duration	6:00
Hydraulic Timestep	1:00
Quality Timestep	0:05
Pattern Timestep	1:00
Pattern Start	0:00
Report Timestep	1:00
Report Start	0:00
Start ClockTime	0:00:00
Statistic	NONE

[REPORT]

Status	No
Summary	No
Page	0

[OPTIONS]

Units	GPM
-------	-----

Headloss	H-W
Specific Gravity	1.000000
Viscosity	1.000000
Trials	40
Accuracy	0.01000000
CHECKFREQ	2
MAXCHECK	10
DAMPLIMIT	0
Unbalanced	Continue 10
Pattern	Day1
Demand Multiplier	1.0000
Emitter Exponent	0.5000
Quality	NONE mg/L
Diffusivity	1.000000
Tolerance	0.10000000

[COORDINATES]

;Node	X-Coord	Y-Coord
LH1	2665.44	6958.05
LH2	2622.42	6061.26
LH3	2623.14	5234.82
HS1	2625.78	3440.61
HS2	3544.26	3268.58
HS3	4408.73	3237.66
DEM	5947.19	5396.10
1	2474.32	6523.97
2	3107.88	3441.78
Well1	875.58	8986.18
Well2	875.58	8740.40
Well3	875.58	8494.62
Well4	875.58	8233.49
Well5	875.58	7926.27
Well6	875.58	7649.77
ElevTank	5499.43	5396.10

[VERTICES]

;Link	X-Coord	Y-Coord
ClearWellIn	2479.13	6952.48
ClearWellOut	2479.13	6060.91
Distribution	5947.19	3237.66
WellPump1	2672.81	8970.81
WellPump2	2672.81	8740.40

WellPump3	2672.81	8479.26
WellPump4	2672.81	8233.49
WellPump5	2672.81	7926.27
WellPump6	2672.81	7665.13
LHP1	2394.45	5861.68
LHP1	2399.43	5408.42
LHP2	2553.84	5866.66
LHP2	2554.41	5383.73
LHP3	2703.27	5861.68
LHP3	2703.32	5383.73
LHP4	2852.69	5866.66
LHP4	2852.23	5383.73
HSP1	3544.75	3536.08
HSP1	4412.61	3536.08
HSP2	3544.75	3382.93
HSP2	4412.61	3380.71
HSP3	3544.75	3140.99
HSP3	4412.61	3140.99
HSP4	3544.75	2983.40
HSP4	4413.08	2981.02
[LABELS]		
;X-Coord	Y-Coord	Label & Anchor Node
2953.41	6966.10	"Low head pump station"
2661.56	2834.00	"High service pump station"
1059.91	9431.64	"Well field"
6040.97	5076.70	"Distribution system"
2815.17	4600.51	"Transmission line"
[END]		

APPENDIX B

MATLAB FUNCTIONS

In general terms, the `cityWdsRun.m` function serves as the core EPANET simulation function for all other functions. It calls functions from the EPANET Toolkit DLL function library in order to run hydraulic calculations on the EPANET INP network file. `cityWdsRun.m` also calls the function `errCk.m` to evaluate necessary penalties for errors returned by EPANET.

The calibration stage is executed from the function `calibrateGA.m`. It repeatedly calls the function `cityWdsCal.m`, which serves as a wrapper function to `cityWdsRun.m` for calibration purposes. The next function, `calibrateMonte.m`, also repeatedly calls the function `cityWdsCal.m` while varying the input parameters.

Enumeration of efficient pump combinations is carried out by the function `enumPumps.m`. It is a standalone function that only requires the output variables from the calibration stage.

The optimization stage is executed from the function `optimizeGA.m`. It repeatedly calls the function `cityWdsOpt.m`, which serves as a wrapper function to `cityWdsRun.m` for optimization purposes.

B.1. cityWdsRun.m

```
% MAIN FUNCTION
function [tankError,p,obsTsHours,obsTankHead,tankHead,pumpStats,dem] = ...
    cityWdsRun(scen,inpt,pumpParam,optForecast,optPumpFlow,optTankLvls,forceDuration)

% Set the flag "I'm optimizing right now" variable
if exist('optTankLvls','var')
    if ~isempty(optTankLvls)
        optimizing=1;
    else
        optimizing=0;
    end
else
    optimizing=0;
end

global shouldGraph
global opPtsFinal
global lhpChar lhpEff hsp1Char hsp1Eff hsp2Char hsp2Eff well3Char well3Eff well5Char well5Eff

shouldSaveInpFile = 0; % This will save each scenario's inp file after simulation
shouldSaveReport = 0;
statusReportLevel = 0;
tankLinkStatus = 1;
p = 0; % Warning code penalty
tankError=0;

% dz limits for optimization - comes from enumPumpsNum.m
dzIncr=1;
dzMin=630;
dzMax=780;

% EPANET variables
inpFile = 'wdsFinal.inp';
if optimizing==0
    duration = 7*24*60*60;
end
if optimizing==1
    duration = max(optPumpFlow(:,1))*60*60;
```

```

end
if exist('forceDuration','var')
    duration = forceDuration*24*60*60;
end
hydTStep = 1*60*60;
%saveFlag = 0;
reportStart = 0;
reportStep = hydTStep;
trials = 40; % default: 40
accuracy = 0.01; % 0.01
tolerance = 0.1; % 0.1
demandMult = 1; % 1
tankDiameter = 150; % Toolkit doesn't work for this, must set manually. Can't calibrate
wellElev = 0; % Arbitrary - all that matters is dz between well and tank

if optimizing==0
    % Minimum tank level
    if length(inpt)>=13
        tankHeadMin = wellElev + inpt(13); % wellElev + dz
    else
        tankHeadMin = 0;
    end

    obsFile={'2009peak2.csv','2009peak.csv','2009rise.csv','2008fall.csv','2007flat.csv'};
    calcColor={'m','c','r','g','b'};
    %obsColor='k';

    % If not calibrating pumpParam, just set some estimated values
    if pumpParam==0
        % Guesses
        switch scen
            case 1
                pumpSetting=[5,3,1,1];
            case 2
                pumpSetting=[5,3,1,1];
            case 3
                pumpSetting=[3,3,1,1];
            case 4
                pumpSetting=[2,3,1,1];
            case 5
                pumpSetting=[1,3,1,0];
        end
    end
end

```

```

else
    % Inputted values for a specific scenario from pumpParam
    if length(pumpParam)>4
        a = 1+(scen-1)*4;
        b = scen*4;
        pumpSetting=pumpParam(a:b);
        clear a, clear b
    else
        pumpSetting=pumpParam;
    end
end
end

end

%% CALIBRATION - LOAD DEMAND PATTERNS AND TANK LEVEL SERIES

if optimizing==0
    % Load previously saved matrix if it exists
    if exist(strcat(obsFile{scen},'.mat'),'file')==2
        load(strcat(obsFile{scen},'.mat'))
    else
        % Otherwise, load observed demands from the .csv
        obsData = csvread(obsFile{scen},1,1);

        % Go from 15min increments (obsData from .csv) to 1hr increments (obsPat)
        % Will eventually divide obsPat by base demand at the end

        % Iterate through weekdays until you get to 1 (i.e. Sunday), THEN start obsPat
        % i is the current position in obsData
        % obsPat(1,1) is the first weekday (1.000)
        i=1;
        obsPat(1,1) = 0;
        while obsPat(1,1) ~= 1
            obsPat(1,:) = [];
            i=i+1;
            obsPat(1,1) = obsData(i,1);
        end
        % obsPat(1,2) is the demand at that time
        obsPat(1,2) = mean([obsData(i-2,2) obsData(i-1,2) obsData(i,2) obsData(i+1,2)]);
        % obsPat(1,3) is the tank volume at that time in gallons
        obsPat(1,3) = mean([obsData(i-2,3) obsData(i-1,3) obsData(i,3) obsData(i+1,3)]);
    end
end

```

```

% j is the current position in obsPat
j=1;
% Start in obsData where weekday=1.000
for k = i:length(obsData)
    % Move through obsData until it's been 1 hour (1/24 days minus a tolerance)
    if obsData(k,1)-obsPat(j,1) >= 1/24-.001
        % Increment obsPat position
        j=j+1;
        % Save weekday from obsData
        obsPat(j,1) = obsData(k,1);
        % Save moving average of demand
        obsPat(j,2) = mean([obsData(k-2,2) obsData(k-1,2) obsData(k,2) obsData(k+1,2)]);
        % Save moving average of tank volume
        obsPat(j,3) = mean([obsData(k-2,3) obsData(k-1,3) obsData(k,3) obsData(k+1,3)]);
    end
end

% Find base demand
baseDem = mean(obsPat(:,2));
% Normalize demands to turn it into a pattern
obsPat(:,2) = obsPat(:,2)/baseDem;

save(strcat(obsFile{scen},'.mat'), 'obsData', 'obsPat', 'baseDem')
end

initTankLevel=tankHeadMin+obsPat(1,3,:)/(3.14159*tankDiameter^2/4*7.48);
end

if optimizing==1
    % Find base demand
    baseDemOpt = mean(optForecast(:,2));
    % Normalize demands to turn it into a pattern
    optForecast(:,2) = optForecast(:,2)/baseDemOpt;
end

%% SIMULATION SETUP

% Initialize pointers for time, timestep, index, generic float
% Note: int32Ptr is equivalent to long or int in C, singlePtr is float
idx = 0;
f = 0;
t = 0;

```

```

dt = hydTStep;

pt = libpointer('int32Ptr',t);
pdt = libpointer('int32Ptr',dt);
pidx = libpointer('int32Ptr',idx);
pf = libpointer('singlePtr',f);

% Load library and open .inp file
if ~libisloaded('epanet2'), loadlibrary('epanet2', 'epanet2.h'); end
wdsRpt = strcat('zoutput',int2str(scen),'.rpt');
wdsOut = strcat('zoutput','.bin');
%wdsHyd = strcat('zoutput','.hyd');

% For all EPANET calls in this program, e is the error code for one
% operation and is checked by errCk.m. p is the running total of GA
% penalties for the entire simulation.
e=calllib('epanet2', 'ENopen', inFile, wdsRpt, wdsOut); p=p+errCk(e,pt);

% Set up parameters
e=calllib('epanet2', 'ENsettimeparam', 0, duration); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsettimeparam', 1, hydTStep); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsettimeparam', 6, reportStart); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsettimeparam', 5, reportStep); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetoption', 0, trials); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetoption', 1, accuracy); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetoption', 2, tolerance); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetoption', 4, demandMult); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetstatusreport', statusReportLevel); p=p+errCk(e,pt);

% Get indices of pipes and nodes
e=calllib('epanet2', 'ENgetlinkindex', 'ClearWellIn', pidx); p=p+errCk(e,pt);
idxClearWellIn = get(pidx,'value');
e=calllib('epanet2', 'ENgetlinkindex', 'ClearWellOut', pidx); p=p+errCk(e,pt);
idxClearWellOut = get(pidx,'value');
e=calllib('epanet2', 'ENgetlinkindex', 'Transmission', pidx); p=p+errCk(e,pt);
idxTransmission = get(pidx,'value');
e=calllib('epanet2', 'ENgetlinkindex', 'GroundTankIn', pidx); p=p+errCk(e,pt);
idxGroundTankIn = get(pidx,'value');
e=calllib('epanet2', 'ENgetlinkindex', 'GroundTankOut', pidx); p=p+errCk(e,pt);
idxGroundTankOut = get(pidx,'value');
e=calllib('epanet2', 'ENgetlinkindex', 'Distribution', pidx); p=p+errCk(e,pt);
idxDist = get(pidx,'value');

```

```

e=calllib('epanet2', 'ENgetlinkindex', 'Distribution2', pidx); p=p+errCk(e,pt);
idxDist2 = get(pidx,'value');
e=calllib('epanet2', 'ENgetnodeindex', 'ElevTank', pidx); p=p+errCk(e,pt);
idxTank = get(pidx,'value');
e=calllib('epanet2', 'ENgetnodeindex', 'DEM', pidx); p=p+errCk(e,pt);
idxDEM = get(pidx,'value');

% Get well and well pump indices
idxWell=zeros(1,6);
idxWP=zeros(1,6);
for i=1:6
    e=calllib('epanet2', 'ENgetnodeindex', strcat('Well',int2str(i)), pidx); p=p+errCk(e,pt);
    idxWell(i) = get(pidx,'value');
    e=calllib('epanet2', 'ENgetlinkindex', strcat('WellPump',int2str(i)), pidx); p=p+errCk(e,pt);
    idxWP(i) = get(pidx,'value');
end
% Get other pump indices
idxLHP=zeros(1,4);
idxHSP=zeros(1,4);
for i=1:4
    e=calllib('epanet2', 'ENgetlinkindex', strcat('LHP',int2str(i)), pidx); p=p+errCk(e,pt);
    idxLHP(i) = get(pidx,'value');
    e=calllib('epanet2', 'ENgetlinkindex', strcat('HSP',int2str(i)), pidx); p=p+errCk(e,pt);
    idxHSP(i) = get(pidx,'value');
end
idxPumps = [idxWP idxLHP idxHSP];

% Set well elevations
for i=1:6
    e=calllib('epanet2', 'ENsetnodevalue', idxWell(i), 0, wellElev); p=p+errCk(e,pt);
end

% Set link parameters
e=calllib('epanet2', 'ENsetlinkvalue', idxClearWellIn, 0, inpt(1)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxClearWellIn, 1, inpt(2)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxClearWellIn, 2, inpt(3)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxClearWellOut, 0, inpt(1)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxClearWellOut, 1, 1); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxClearWellOut, 2, inpt(3)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxTransmission, 0, inpt(4)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxTransmission, 1, inpt(5)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxTransmission, 2, inpt(6)); p=p+errCk(e,pt);

```

```

e=calllib('epanet2', 'ENsetlinkvalue', idxGroundTankIn, 0, inpt(4)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxGroundTankIn, 1, 1); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxGroundTankIn, 2, inpt(6)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxGroundTankOut, 0, inpt(4)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxGroundTankOut, 1, 1); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxGroundTankOut, 2, inpt(6)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist, 0, inpt(7)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist, 1, inpt(8)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist, 2, inpt(9)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist2, 0, inpt(10)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist2, 1, inpt(11)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist2, 2, inpt(12)); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENsetlinkvalue', idxDist2, 11, tankLinkStatus); p=p+errCk(e,pt);

if optimizing==0
    % Set initial tank level
    e=calllib('epanet2', 'ENsetnodevalue', idxTank, 8, initTankLevel); p=p+errCk(e,pt);
    % Apply base demand to DEM node
    e=calllib('epanet2', 'ENsetnodevalue', idxDEM, 1, baseDem); p=p+errCk(e,pt);

    % Break patterns in [obs_pat] out into 7 day periods, save them in the .inp
    for i=1:7
        % Save obsPat(1:24) to pattern 1 i.e. day 1, obsPat(25:48) to pattern 2, etc.
        e=calllib('epanet2', 'ENsetpattern', i, obsPat((i-1)*24+1:i*24,2)',24); p=p+errCk(e,pt);
    end
end

if optimizing==1
    % Set initial tank level
    e=calllib('epanet2', 'ENsetnodevalue', idxTank, 8, optTankLvls(1)); p=p+errCk(e,pt);
    % Apply base demand to DEM node
    e=calllib('epanet2', 'ENsetnodevalue', idxDEM, 1, baseDemOpt); p=p+errCk(e,pt);

    % Break patterns in hourly forecasted flows out into day-long periods, save them in the .inp
    for i=1:length(optForecast(:,2))/24
        % Save obsPat(1:24) to pattern 1 i.e. day 1, obsPat(25:48) to pattern 2, etc.
        if i<=optForecast(length(optForecast(:,2))) % make sure flows provided go to the end of this day
            e=calllib('epanet2', 'ENsetpattern', i, optForecast((i-1)*24+1:i*24,2)',24); p=p+errCk(e,pt);
        else
            disp('WARNING: forecasted flows stop before the end of a full day')
        end
    end
end
end

```

```

    % Warn me if I forgot to include 0 as the first pump flow time
    if optPumpFlow(1,1) > 0
        disp('WARNING: first target pump flow time is not zero')
    end
end

%% BEGIN SIMULATION

e=calllib('epanet2', 'ENopenH'); p=p+errCk(e,pt);
e=calllib('epanet2', 'ENinitH', shouldSaveReport); p=p+errCk(e,pt);

% Initialize other parameters

if optimizing==0

    % 1: Turn on number of well pumps specified in pumpSetting - ranges from 1 to 6 (integer)
    for i=1:6
        if i <= pumpSetting(1)
            e=calllib('epanet2', 'ENsetlinkvalue', idxWP(i), 4, 1); p=p+errCk(e,pt); % on
            e=calllib('epanet2', 'ENsetlinkvalue', idxWP(i), 12, 1); p=p+errCk(e,pt); % full-blast
        else
            e=calllib('epanet2', 'ENsetlinkvalue', idxWP(i), 4, 0); p=p+errCk(e,pt); % off
        end
    end

    % 2: Low-head pumps (only 1 VFD) - pumpSetting ranges from 0 to 4 (continuous)
    for i=1:4
        if i <= pumpSetting(2)
            e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(i), 4, 1); p=p+errCk(e,pt); % on
            e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(i), 12, 1); p=p+errCk(e,pt); % full-blast
        elseif i < pumpSetting(2)+1
            e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(i), 4, 1); p=p+errCk(e,pt); % on
            e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(i), 12, pumpSetting(2)+1-i); p=p+errCk(e,pt); % speed
        else
            e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(i), 4, 0); p=p+errCk(e,pt); % off
        end
    end

    % 3: Non-vfd high-service pumps - pumpSetting ranges from 0 to 2 (integer)
    for i=1:2
        if i <= pumpSetting(3)

```



```

        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(i), 4, 1); p=p+errCk(e,pt); % on
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(i), 12, 1); p=p+errCk(e,pt); % full-blast
    else
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(i), 4, 0); p=p+errCk(e,pt); % off
    end
end

% 4: VFD high-service pumps - pumpSetting ranges from 0 to 1 (continuous)
% Assume both VFDs run at same speed
for i=1:2
    if pumpSetting(4) == 0
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(i+2), 4, 0); p=p+errCk(e,pt); % off
    else
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(i+2), 4, 1); p=p+errCk(e,pt); % on
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(i+2), 12, pumpSetting(4)/2); p=p+errCk(e,pt); % speed
    end
end

end

%% BEGIN SIMULATION LOOP

i=1;

% Initialize growing arrays to minimum size for speed
tSecs = zeros(1,duration/hydTStep);
tankHead = zeros(1,duration/hydTStep);
dem = zeros(1,duration/hydTStep);
pumpStats = zeros(duration/hydTStep,length(idxPumps),5);
pumpSetIdx = 1;

% Execute simulation until dt goes to 0 (i.e. simulation time is completed)
while (get(pdt,'value') > 0) && p<1e30

    if optimizing==0
        % Get pattern of demand node
        e=calllib('epanet2', 'ENgetnodevalue', idxDEM, 2, pf); p=p+errCk(e,pt);

        % What day is this? If the pattern doesn't match, change it
        day = floor(double(t)/60/60/24)+1;
        if get(pf,'value') ~= day
            e=calllib('epanet2', 'ENsetnodevalue', idxDEM, 2, day); p=p+errCk(e,pt);
        end
    end
end

```

```

    end
end

if optimizing==1

    % Get pattern of demand node
    e=calllib('epanet2', 'ENgetnodevalue', idxDEM, 2, pf); p=p+errCk(e,pt);

    % What day is this? If the pattern doesn't match, change it
    day = floor(double(t)/60/60/24)+1;
    if get(pf,'value') ~= day
        e=calllib('epanet2', 'ENsetnodevalue', idxDEM, 2, day); p=p+errCk(e,pt);
    end

    % Change pump configuration, but only if I *just* crossed over
    % into the next optPumpFlow time.
    if get(pt,'value')/3600 >= optPumpFlow(pumpSetIdx,1)

        e=calllib('epanet2', 'ENgetnodevalue', idxWell(1), 0, pf);
        p=p+errCk(e,pt);
        wellLvl=get(pf,'value'); % Water surface level in well
        e=calllib('epanet2', 'ENgetnodevalue', idxTank, 8, pf);
        p=p+errCk(e,pt);
        tankLvl=get(pf,'value'); % Tank head
        e=calllib('epanet2', 'ENgetnodevalue', idxTank, 0, pf);
        p=p+errCk(e,pt);
        tankLvl=tankLvl+get(pf,'value'); % Add elevation of tank base

        % Get dz
        dz=tankLvl-wellLvl;
        if dz<dzMin
            disp('ERROR: dz<dzMin! Make a wider range of dzs in opPtsFinal.')
        end
        if dz>dzMax
            disp('ERROR: dz>dzMax! Make a wider range of dzs in opPtsFinal.')
        end
        dzIdx=round((dz-dzMin)/dzIncr)+1; % index for lookup in opPtsFinal

        % Round given target pump flow to limits of flows in optPumpFlow
        % Otherwise, 'nearest' interpolation used below won't work
        % (Note: opPtsFinal must be defined as a global variable!)
        if optPumpFlow(pumpSetIdx,2) < opPtsFinal(dzIdx,1,1)

```

```

    % Round to min flow
    if opPtsFinal(dzIdx,1,1)-optPumpFlow(pumpSetIdx,2)<optPumpFlow(pumpSetIdx,2)
        optPumpFlow(pumpSetIdx,2) = opPtsFinal(dzIdx,1,1);
    else
        optPumpFlow(pumpSetIdx,2)=0;
    end
elseif optPumpFlow(pumpSetIdx,2) > max(opPtsFinal(dzIdx,:,1))
    % Round to max flow
    optPumpFlow(pumpSetIdx,2) = max(opPtsFinal(dzIdx,:,1));
end

% Grab high-eff. pump combo corresponding to the target pumped Q
if optPumpFlow(pumpSetIdx,2)==0
    for z=1:4
        pumpSetting(pumpSetIdx,z)=0;
    end
else
    [opPtsFinalFlow, uniqueIdx] = unique(opPtsFinal(dzIdx,:,1));
    opPtsFinalConfig=zeros(length(uniqueIdx),4);
    for z=1:4
        opPtsFinalConfig(:,z)=opPtsFinal(dzIdx,uniqueIdx,z+3);
        pumpSetting(pumpSetIdx,z)=interp1(opPtsFinalFlow,...
            opPtsFinalConfig(:,z),optPumpFlow(pumpSetIdx,2),'nearest');
    end
end

% 1: Turn on number of well pumps in pumpSetting; goes from 1-6 (integer)
for j=1:6
    if j <= pumpSetting(pumpSetIdx,1)
        % On
        e=calllib('epanet2', 'ENsetlinkvalue', idxWP(j), 11, 1);
        p=p+errCk(e,pt);
        % Full-blast
        e=calllib('epanet2', 'ENsetlinkvalue', idxWP(j), 12, 1);
        p=p+errCk(e,pt);
    else
        % Off
        e=calllib('epanet2', 'ENsetlinkvalue', idxWP(j), 11, 0);
        p=p+errCk(e,pt);
    end
end
end

```

```

% 2: Low-head pumps (1 VFD); pumpSetting goes from 0-4 (continuous)
for j=1:4
    if j <= pumpSetting(pumpSetIdx,2)
        % On
        e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(j),...
            11, 1);
        p=p+errCk(e,pt);
        % Full-blast
        e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(j),...
            12, 1);
        p=p+errCk(e,pt);
    elseif j < pumpSetting(pumpSetIdx,2)+1
        % On
        e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(j),...
            11, 1);
        p=p+errCk(e,pt);
        % Set speed
        e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(j),...
            12, pumpSetting(pumpSetIdx,2)+1-j);
        p=p+errCk(e,pt);
    else
        % Off
        e=calllib('epanet2', 'ENsetlinkvalue', idxLHP(j),...
            11, 0); p=p+errCk(e,pt);
    end
end

% 3: Non-VFD high-service pumps; pumpSetting goes from 0-2 (integer)
for j=1:2
    if j <= pumpSetting(pumpSetIdx,3)
        % On
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(j),...
            11, 1);
        p=p+errCk(e,pt);
        % Full-blast
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(j),...
            12, 1);
        p=p+errCk(e,pt);
    else
        % Off
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(j),...
            11, 0);
    end
end

```

```

        p=p+errCk(e,pt);
    end
end

% 4: VFD high-service pumps; pumpSetting goes from 0-1 (continuous)
% Assume both VFDs run at same speed
for j=1:2
    if pumpSetting(pumpSetIdx,4) == 0
        % Off
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(j+2),...
            11, 0); p=p+errCk(e,pt);
    else
        % On
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(j+2),...
            11, 1); p=p+errCk(e,pt);
        % Set speed
        e=calllib('epanet2', 'ENsetlinkvalue', idxHSP(j+2),...
            12, pumpSetting(pumpSetIdx,4));
        p=p+errCk(e,pt);
    end
end
% Iterate pumpSetIdx; I've taken care of this optPumpFlow timestep
pumpSetIdx=pumpSetIdx+1;
end
end

% EXECUTE SIMULATION
e=calllib('epanet2', 'ENrunH', pt);
p=p+errCk(e,pt);
e=calllib('epanet2', 'ENnextH', pdt);
p=p+errCk(e,pt);

% Update time and save t/dt values for later
t = get(pt,'value');
tSecs(i) = t;

% Update time series of demand applied to DEM node
e=calllib('epanet2', 'ENgetnodevalue', idxDEM, 9, pf); p=p+errCk(e,pt);
dem(i)=get(pf,'value');

```

```

% Update time series of tank head
e=calllib('epanet2', 'ENgetnodevalue', idxTank, 10, pf); p=p+errCk(e,pt);
tankHead(i)=get(pf,'value');
% Subtract elevation from tank head
e=calllib('epanet2', 'ENgetnodevalue', idxTank, 0, pf); p=p+errCk(e,pt);
tankHead(i)=tankHead(i)-get(pf,'value');

% Get pump stats
for j=1:14
    % Setting/speed
    e=calllib('epanet2', 'ENgetlinkvalue', idxPumps(j), 11, pf); p=p+errCk(e,pt);
    if get(pf,'value')>0
        e=calllib('epanet2', 'ENgetlinkvalue', idxPumps(j), 12, pf); p=p+errCk(e,pt);
        pumpStats(i,j,1) = get(pf,'value');
    else
        pumpStats(i,j,1) = 0;
    end
    % Flow
    e=calllib('epanet2', 'ENgetlinkvalue', idxPumps(j), 8, pf); p=p+errCk(e,pt);
    pumpStats(i,j,2) = get(pf,'value');
    % Head "loss" (should be negative)
    e=calllib('epanet2', 'ENgetlinkvalue', idxPumps(j), 10, pf); p=p+errCk(e,pt);
    pumpStats(i,j,3) = get(pf,'value');
    % Power expended in kilowatts
    e=calllib('epanet2', 'ENgetlinkvalue', idxPumps(j), 13, pf); p=p+errCk(e,pt);
    pumpStats(i,j,4) = get(pf,'value');
    % Energy expended in kilowatt-hours for this timestep
    pumpStats(i,j,5) =pumpStats(i,j,4)*get(pdt,'value')/3600;
end
% Efficiency in percent: efficiency = efficiency_function(flow/speed)
pumpStats(i,1,6)= well3Eff(pumpStats(i,1,2) /pumpStats(i,1,1));
pumpStats(i,2,6)= well5Eff(pumpStats(i,2,2) /pumpStats(i,2,1));
pumpStats(i,3,6)= well3Eff(pumpStats(i,3,2) /pumpStats(i,3,1));
pumpStats(i,4,6)= well5Eff(pumpStats(i,4,2) /pumpStats(i,4,1));
pumpStats(i,5,6)= well3Eff(pumpStats(i,5,2) /pumpStats(i,5,1));
pumpStats(i,6,6)= well5Eff(pumpStats(i,6,2) /pumpStats(i,6,1));
pumpStats(i,7,6)= lhpEff( pumpStats(i,7,2) /pumpStats(i,7,1));
pumpStats(i,8,6)= lhpEff( pumpStats(i,8,2) /pumpStats(i,8,1));
pumpStats(i,9,6)= lhpEff( pumpStats(i,9,2) /pumpStats(i,9,1));
pumpStats(i,10,6)=lhpEff( pumpStats(i,10,2)/pumpStats(i,10,1));
pumpStats(i,11,6)=hsp1Eff( pumpStats(i,11,2)/pumpStats(i,11,1));
pumpStats(i,12,6)=hsp2Eff( pumpStats(i,12,2)/pumpStats(i,12,1));

```

```

    pumpStats(i,13,6)=hsp1Eff( pumpStats(i,13,2)/pumpStats(i,13,1));
    pumpStats(i,14,6)=hsp2Eff( pumpStats(i,14,2)/pumpStats(i,14,1));

    % Iterate array counter
    i=i+1;
end

%% SAVE, WRITE REPORT, END

if shouldSaveInpFile == 1
    e=calllib('epanet2', 'ENsaveinpfile', strcat('wdsFinal',int2str(scen),'.inp')); p=p+errCk(e,pt);
end

if shouldSaveReport == 1
    e=calllib('epanet2', 'ENsaveH'); p=p+errCk(e,pt);
    e=calllib('epanet2', 'ENcloseH'); p=p+errCk(e,pt);
    e=calllib('epanet2', 'ENreport'); p=p+errCk(e,pt);
else
    e=calllib('epanet2', 'ENcloseH'); p=p+errCk(e,pt);
end

% Close and unload library
e=calllib('epanet2', 'ENclose'); p=p+errCk(e,pt);
if libisloaded('epanet2'), unloadlibrary('epanet2'); end

%% RETURN CALIBRATION RESULTS, GRAPHS
tsHours = double(tSecs)/60/60;

if optimizing==0 && p<1e30
    obsTsHours=(obsPat(:,1)-1).*24;
    obsTank=obsPat(:,3);

    % Calculate tank head series and interpolate
    tankHead2 = interp1(tsHours,tankHead,obsTsHours);
    obsTankHead = obsTank/(3.14159*tankDiameter^2/4*7.48)+tankHeadMin;

    % Check plots to ensure patterns are being applied properly
    % obs_dem=obs_pat(:,2).*base_dem;
    % figure
    % hold on
    % plot(ts_day, dem, '-r','linewidth',2)

```

```

% plot(obs_ts, obs_dem, '--b')
% legend('applied demand','observed demand')
% axis([0 7*24 0 30000]);

% Compare observed tank level to calculated levels
tankErrorPt = zeros(1,length(obsTsHours)-1);
tankError = 0;
for i = 1:length(obsTsHours)-1
    tankErrorPt(i) = (tankHead2(i)+tankHead2(i+1))/2 - (obsTankHead(i)+obsTankHead(i+1))/2;
    if ~isnan(tankErrorPt(i))
        tankError = tankError + tankErrorPt(i)^2*(obsTsHours(i+1)-obsTsHours(i));
    end
end

% Make vectors same length
tankErrorPt(i+1)=tankErrorPt(i);

% Apply a penalty if the tank is stagnant (this should never actually happen)
if max(tankHead2)-min(tankHead2) <= 1
    bigPenalty = 1e10/(max(tankHead2)-min(tankHead2));
    tankError = tankError + bigPenalty;
    fprintf('Tank is stagnant; penalty of %10.0f applied\n',bigPenalty)
end

% Plot results
if shouldGraph==1
    hold on
    figure(1);
    plot(obsTsHours, tankErrorPt, '-', 'linewidth',1, 'color',calcColor{scen})
    axis([0 7*24 -50 50]);
    pause(0.001)

%     figure(2);
%     plot(obsTsHours, tankHead2, '-', 'linewidth',1, 'color',calcColor{scen})
%     plot(obsTsHours, obsTankHead, '--', 'color',calcColor{scen})
%     legend('calc tank level','obs tank level')
%     axis([0 7*24 0 200]);
end
elseif optimizing==0
    tankError=9999;
    obsTsHours=[0 0];
    obsTankHead=[0 0];

```



```

end

%% APPLY OPTIMIZATION PENALTIES, PLOT STUFF
if optimizing==1
    % Only add extra penalty for going UNDER the target ending tank level.
    % (If I overshoot the target ending tank level, no extra penalty needed.)
    % Encourages a "top-down" approach.
    if tankHead(length(tankHead))<optTankLvls(2)
        p = p + 1000*abs(optTankLvls(2) - tankHead(length(tankHead)))^2;
    end

    % Add penalty for going under the absolute minimum tank level
    for j=1:length(tankHead)
        if tankHead(j)<optTankLvls(3)
            p = p + 10000*(optTankLvls(3) - tankHead(j))^2;
        end
        %if tankHead(j)<optTankLvls(3)
        %    p = p + (175 - tankHead(j))^2;
        %end
    end

    plot(tsHours,tankHead)

    % Assign unused variables
    obsTsHours=0;
    obsTankHead=0;
end

```

B.2. errCk.m

```
% Check for EPANET errors and return an appropriate penalty for the GA
function penalty = errCk(err,pt)
global monteCarlo
penalty=0;
global suppressWarning
global suppressAnnoyingWarning
suppressError=0;
if monteCarlo==0
    if ~exist('suppressWarning','var')
        suppressWarning=0;
    end
    if ~exist('suppressAnnoyingWarning','var')
        suppressAnnoyingWarning=0;
    end
    if err > 6
        if suppressError==0
            fprintf('ERROR CODE %i at t=%li\n',err,get(pt,'value'))
        end
    end
    if err == 1
        if suppressWarning==0
            fprintf('Warning: System unbalanced at t=%li\n',get(pt,'value'))
        end
        penalty = penalty+1e30;
    end
    if err == 2
        if suppressWarning==0
            fprintf('Warning: System UNSTABLE at t=%li\n',get(pt,'value'))
        end
        penalty = penalty + 1e20;
    end
    if err == 3
        if suppressWarning==0
            fprintf('Warning: System DISCONNECTED at t=%li\n',get(pt,'value'))
        end
        penalty = penalty + 1e30;
    end
    if err == 4
```

```

        if suppressAnnoyingWarning==0
            fprintf('Warning: Pumps cannot deliver flow/head at t=%li\n',get(pt,'value'))
        end
        if penalty==0
            penalty=30000;
        end
        penalty = penalty*1.1;
    end
    if err == 6
        if suppressAnnoyingWarning==0
            fprintf('Warning: Negative pressures at t=%li\n',get(pt,'value'))
        end
        penalty = penalty + 5e2;
    end
    % small effort to speed up Monte Carlo evaluations
elseif monteCarlo==1
    penalty=0;
end
end

```

B.3. calibrateGA.m

```
format compact; format short g;
warning off MATLAB:loadlibrary:typenotfound

%% PREPARE GA -- SYSTEM PARAMETERS

global shouldGraph
shouldGraph = 1;
global well13Eff well15Eff lhpEff hsp1Eff hsp2Eff %#ok<NUSED>
load fittedcurves.mat

global suppressWarning
global suppressAnnoyingWarning
suppressWarning=1;
suppressAnnoyingWarning=1;

for scens=1:5
% lower and upper bounds for constrained minimization
lb(scens,:) = [18 1000 80,... % Diameter, length, roughness
    18 1000 80,...
    18 1000 80,...
    18 1 80,...
    150,... % dz between Well elevation and Minimum tank head
    4 2 0 0,... % Pump parameters
    1;
ub(scens,:) = [36 10000 160,... % Diameter, length, roughness
    60 50000 160,...
    60 100000 160,...
    60 100000 160,...
    700,... % dz between Well elevation and Minimum tank head
    6 4 2 1,... % Pump parameters
    1;
end
dvs = length(lb);

% restrict bounds a little tighter around pump variables
lb(2,14:17)=[4 2 0 0];
ub(2,14:17)=[6 4 2 1];

lb(3,14:17)=[2 1 0 0];
```

```

ub(3,14:17)=[6 4 2 1];

lb(4,14:17)=[2 1 0 0];
ub(4,14:17)=[6 4 2 1];

lb(5,14:17)=[1 0 0 0];
ub(5,14:17)=[4 3 1 1];

popSize = 100;
gens = 30;
trials = 5;
crossRate = 0.7; % part of new population generated by crossover; remainder generated by mutation

for scens = 1:5
    %% EXECUTE GA -- ONE SCENARIO AT A TIME

    tic
    for j = 1:trials
        options = gaoptimset('PopulationSize',popSize);
        options = gaoptimset(options, 'Generations',gens);
        options = gaoptimset(options, 'CreationFcn',@gacreationlinearfeasible);
        options = gaoptimset(options, 'CrossoverFcn',@crossoverarithmic);
        options = gaoptimset(options, 'CrossoverFraction',crossRate);
        options = gaoptimset(options, 'MutationFcn',@mutationadaptfeasible);
        options = gaoptimset(options, 'EliteCount',3);
        %options = gaoptimset(options, 'TolFun',1e3);
        options = gaoptimset(options, 'StallGenLimit',5);
        options = gaoptimset(options, 'PopInitRange',[lb(scens,:); ub(scens,:)]);
        options = gaoptimset(options, 'Display','iter');
        options = gaoptimset(options, 'PlotFcns', @gaplotbestf);

        % EXECUTE GA
        [x(scens,j,:) y(scens,j) exitflag(scens,j)...
         output(scens,j) population(scens,j,:)] = ...
            ga(@(inpt)cityWdsCal(scens,inpt),dvs,' ',' ',' ',...
             lb(scens,:),ub(scens,),' ',options);
    end
    toc

    % save plots and variables
    figs=get(0,'children');
    saveas(figs(1),['calibrationtest_fig1_trial' num2str(j) '.fig'])
    saveas(figs(2),['calibrationtest_fig2_trial' num2str(j) '.fig'])

```

```

        save calibration_resultslc.mat
    end

    % Find the best GA result out of the trials
    [bestTrial, bestIndex] = min(y(scens,:));
    bestX(scens,:) = x(scens,bestIndex,:)

    % Calculate error from the best GA result
    shouldGraph = 1;
    lowestY(scens) = cityWdsCal(scens,bestX(scens,:))

end

%% GENERATE ONE BESTX TO USE WITH NEXT GA

% fineTuneX is an aggregate of the 5 best decision variable sets to be
% applied to all scenarios simultaneously. It's built from the bestX
% values just calibrated for each scenario individually. Then the pump
% params are "filled out" using the best parameters from all scenarios.
fineTuneX = bestX;

for scens=1:5
    for i=1:5
        % Populate pump parameters
        fineTuneX(i,14+4*(scens-1):14+4*scens-1) = bestX(scens,14:17);

        % Round number of no-vfd well pumps and HSPs
        fineTuneX(i,14+4*(scens-1)) = round(fineTuneX(i,14+4*(scens-1)));
        fineTuneX(i,16+4*(scens-1)) = round(fineTuneX(i,16+4*(scens-1)));
    end
end

save calibration_resultslc.mat

%% EXECUTE GA AGAIN -- NOW ALL AT ONCE

% Use bestX for each individual scenario as starting point for 5 separate
% GA runs... calibrating all scenarios simultaneously.

load calibration_resultslc.mat

shouldGraph = 0;

```

```

tic
close all

dvs2=length(fineTuneX);

lb2 = [lb(1,1:13), 1 0 0 0, 1 0 0 0, 1 0 0 0, 1 0 0 0, 1 0 0 0];
ub2 = [ub(1,1:13), 6 4 2 1, 6 4 2 1, 6 4 2 1, 6 4 2 1, 6 4 2 1];

popSize2 = 100;
gens2 = 50;
crossRate2 = 0.4; % part of new population generated by crossover; remainder generated by mutation
mutRate2 = 0.2; % probability that any single mutated variable will change

options2 = options;
options2 = gaoptimset(options2, 'PopulationSize',popSize2);
options2 = gaoptimset(options2, 'Generations',gens2);
options2 = gaoptimset(options2, 'PopInitRange',[lb2; ub2]);
options2 = gaoptimset(options2, 'CrossoverFcn',@crossoverarithmetic);
options2 = gaoptimset(options2, 'CrossoverFraction',crossRate2);
options2 = gaoptimset(options2, 'MutationFcn', {@mutationuniform, mutRate2});

for scens = 2:5
    % Fill the population with solutions from step 1
    options2 = gaoptimset(options2,'InitialPopulation', repmat(fineTuneX(scens,:),popSize2,1));

    % EXECUTE GA
    [x2(scens,:) y2(scens) exitflag2(scens) output2(scens)] = ...
        ga(@inpt)cityWdsCal([1 5],inpt),dvs2,[],[],[],lb2,ub2,[],options2);
    toc

    save calibration_results2c.mat

    % save GA plot
    figs2=get(0,'children');
    for f=1:length(figs2)
        saveas(figs2(f),['calibrationtestpt2_fig' num2str(f) '_scen' num2str(scens) '.fig'])
    end
    close(figs2)

    % save cityWdsRun plot
    shouldGraph=1;
    testX = cityWdsCal([1 5],x2(scens,:))

```

```

    figs2=get(0,'children');
    for f=1:length(figs2)
        saveas(figs2(f),['calibrationtestpt2_fig' num2str(f) '_scen' num2str(scens) '.fig'])
    end
    close(figs2)
    shouldGraph=0;

    save calibration_results2c.mat
end

% Find the best GA result out of the trials
[bestTrial2, bestIndex2] = min(y2);
bestX2 = x2(bestIndex2,:);

% Calculate error from the best GA result
shouldGraph = 1;

lowestY2 = cityWdsCal([1 5],bestX2)

save calibration_results2c.mat

%% LAST PART - CALIBRATE PUMP PARAMETERS ONLY

load calibration_results2c.mat

shouldGraph = 0;
tic
close all

pipeParam = bestX2(1:13);

dvs3 = 4; % pump parameters per scenario

lb3 = [ 1 0 0 0 ];
ub3 = [ 6 4 2 1 ];

popSize3 = 100;
gens3 = 50;
crossRate3 = 0.5; % part of new population generated by crossover; remainder generated by mutation
mutRate3 = 0.3; % probability that any single mutated variable will change

```



```

options3 = options2;
options3 = gaoptimset(options3, 'PopulationSize', popSize3);
options3 = gaoptimset(options3, 'Generations', gens3);
options3 = gaoptimset(options3, 'PopInitRange', [lb3; ub3]);
options3 = gaoptimset(options3, 'CrossoverFcn', @crossoverarithmetic);
options3 = gaoptimset(options3, 'CrossoverFraction', crossRate2);
options3 = gaoptimset(options3, 'MutationFcn', {@mutationuniform, mutRate3});
options3 = gaoptimset(options3, 'StallGenLimit', 10);
options3 = gaoptimset(options3, 'TolFun', 1e2);

for scens = 5:5
    % Fill the population with pump parameter solutions from step 2
    startIndex3 = 14+(scens-1)*4;
    endIndex3 = 17+(scens-1)*4;
    options3 = gaoptimset(options3, 'InitialPopulation', ...
        repmat(bestX2(startIndex3:endIndex3), popSize3, 1));

    % EXECUTE GA - PUMP PARAMETERS ONLY (PIPE PARAMETERS FROZEN)
    [x3(scens,:) y3(scens) exitflag3(scens) output3(scens)] = ...
        ga(@ (inpt) cityWdsCal(scens, [pipeParam inpt]), dvs3, '', '', '', lb3, ub3, '', options3);
    toc

    save calibration_results3c.mat

    % save GA plot
    figs3=get(0, 'children');
    for f=1:length(figs3)
        saveas(figs3(f), ['calibrationtestpt3a_fig' num2str(f) '_scen' num2str(scens) '.fig'])
    end
    close(figs3)

    % save cityWdsRun plot
    shouldGraph=1;
    testX = cityWdsCal(scens, [pipeParam x3(scens,:)])
    figs3=get(0, 'children');
    for f=1:length(figs3)
        saveas(figs3(f), ['calibrationtestpt3b_fig' num2str(f) '_scen' num2str(scens) '.fig'])
    end
    close(figs3)
    shouldGraph=0;

    save calibration_results3c.mat

```

```

end

% build final calibration parameter results
bestX3 = bestX2(1:13);
bestX3(14:17) = x3(1,:) % from scenario 1
bestX3(18:21) = x3(2,:) % 2
bestX3(22:25) = x3(3,:) % 3
bestX3(26:29) = x3(4,:) % 4
bestX3(30:33) = x3(5,:) % 5

% Calculate error from the best GA results so far
shouldGraph = 1;

lowestY3 = cityWdsCal([1 5],bestX3)
for sc=1:5
    [tankError{sc}, p{sc}, obsTsHours{sc}, obsTankHead{sc}, tankHead{sc}, pumpStats{sc}] =
cityWdsCal(sc,bestX3b);
end

for sc=1:5
    hold on
    plot(obsTsHours{sc}{sc}, (tankHead{sc}{sc}(1:168)-bestX3(13))*(3.14159*150^2/4*7.48), '-
','linewidth',2,'color','k')
    plot(obsTsHours{sc}{sc}, (obsTankHead{sc}{sc}-bestX3(13))*(3.14159*150^2/4*7.48),
':','linewidth',2,'color','k')
    axis([0 24*7 0 10e6]);
    legend('Calculated tank level','Observed tank level','Location','Best')
    xlabel('Time (hours)')
    ylabel('Storage volume (gallons)')
    set(gca,'XTick',0:12:24*7)
    set(gca,'YGrid','on')

    saveas(gca,['calibration_tanklevels_scen' num2str(sc) '.fig'])
    saveas(gca,['calibration_tanklevels_scen' num2str(sc) '.png'])
    close all
end

save calibration_results3c.mat

```

B.4. cityWdsCal.m

```
function [tankError, p, obsTsHours, obsTankHead, tankHead, pumpStats] = cityWdsCal(scens,inpt)

% When running GA for calibrating scenarios, inpt is a vector with size 17
% 1,2,3 - ClearWellIn/Out diameter, length, roughness
% 4,5,6 - Transmission and GroundTankIn/Out diameter, length, roughness
% 7,8,9 - Distribution diameter, length, roughness
% 10,11,12 - Distribution2 diameter, length, roughness
% 13 - dz between Well elevation and Minimum tank head
% 14,15,16,17 - pump parameters

% Scenarios:
% 1 - 2009peak2
% 2 - 2009peak
% 3 - 2009rise
% 4 - 2008fall
% 5 - 2007flat

% Initialize variables
%global shouldGraph
global monteCarlo

if isempty(monteCarlo)
    monteCarlo=0;
end
if isempty(scens)
    first = 1;
    last = 5;
else
    first=scens(1);
    last= scens(length(scens));
end
tankError = 0;
pumpInit = 0;
pumpParam = [0 0 0 0];

% Calibrating all parameters, including new pump parameters
if length(inpt)==12+1+4*5 || length(inpt)==12+1+4
```

```

% New inpt parameters - i.e. pumpParam
% 12: number of well pumps on (integer, 1-6)
% 13: number of low-head pumps on, vfd and non-vfd (continuous, 0-4)
% 14: number of non-vfd high service pumps on (integer, 0-2)
% 15: number of vfd high service pumps on (continuous, 0-1)

if length(inpt)==12+1+4*5
    for i=1:4
        inpt(14+4*(i-1)) = round(inpt(14+4*(i-1))); % no-vfd well pumps
        inpt(16+4*(i-1)) = round(inpt(16+4*(i-1))); % no-vfd HSPs
    end
elseif length(inpt)==12+1+4
    inpt(14)=round(inpt(14)); % no-vfd well pumps
    inpt(16)=round(inpt(16)); % no-vfd HSPs
end

for sc=first:last
    % if we are dealing with multiple pump parameters
    if length(inpt)==12+1+4*5
        % load the right pump parameters
        pumpParam = [inpt(14+4*(sc-1)) inpt(14+4*(sc-1)+1)...
                    inpt(14+4*(sc-1)+2) inpt(14+4*(sc-1)+3)];
        % or if we are dealing with only one set of pump parameters
    elseif length(inpt)==12+1+4
        % just load those
        pumpParam = inpt(14:17);
    end

    if monteCarlo==0
        [tankError(sc) p(sc) obsTsHours{sc} ...
         obsTankHead{sc} tankHead{sc} pumpStats{sc}] = ...
            cityWdsRun(sc,inpt(1:13),pumpParam);
    elseif monteCarlo==1
        clear tankError
        % same operation, but variables are saved differently for MC organization
        [ a b c d e f ] = cityWdsRun(sc,inpt(1:13),pumpParam);
        tankError{sc}=a;
        p{sc}=b;
        obsTsHours{sc}=c;
        obsTankHead{sc}=d;
        tankHead{sc}=e;
        pumpStats{sc}=f;
    end
end

```

```

        end
    end
end

% Add penalties together with penalties from warning codes
for sc=first:last
    if monteCarlo==0
        tankError(sc) = tankError(sc)+p(sc);
    elseif monteCarlo==1
        tankError{sc} = tankError{sc}+p{sc};
    end
end
if length(tankError)>1 && monteCarlo==0
    tankError = sum(tankError);
elseif length(tankError)>1 && monteCarlo==1
    g=0;
    for sc=first:last
        g=g+tankError{sc};
    end
    clear tankError
    tankError=g;
end

```

B.5. calibrateMonte.m

```
format compact; format short g; close all;
global suppressWarning
global suppressAnnoyingWarning
suppressWarning=1; suppressAnnoyingWarning=1;
warning off MATLAB:loadlibrary:typenotfound

global lhpEff hsp1Eff hsp2Eff well3Eff well5Eff %#ok<NUSED>
load fittedcurves.mat

global shouldGraph
shouldGraph=0;

global monteCarlo
monteCarlo=1;

%% PREPARE MONTE CARLO RESULTS - RUN **ONCE**

variance_pct = 0.1:0.1:0.4; % target percentage variances in each parameter
mcRuns = 200; % number of Monte Carlo runs
scenarios = 5; % number of week-long periods I'm dealing with
simlength = 7*24; % number of hours in simulation

% best calibrated parameters for scenarios 1-5 - calculated using calibrateGA.m
load calibration_results3c_manual5.mat bestX3b
params = bestX3b;

% calculate new parameters, EXCEPT keep dz the same
params_new=zeros(length(variance_pct),mcRuns,length(params)-1);

tic

for v = 1:length(variance_pct)
    for i = 1:mcRuns
        for j = 1:length(params)
            % generate new params - 2 stdevs away from the mean is within v% of original
            if j~=13
                params_new(v,i,j) = normrnd(params(j),variance_pct(v)*params(j)/2);
            elseif j==13
```

```

        params_new(v,i,j) = params(j);
    end
end
[a b c d e f] = cityWdsCal([1 5],params_new(v,i,:));
tankError{v}{i}=a;
p{v}{i}=b;
obsTsHours{v}{i}=c;
obsTankHead{v}{i}=d;
tankHead{v}{i}=e;
pumpStats{v}{i}=f;
if mod(i,50)==0
    % save results every 50 runs in case my processor melts
    save montecarloresults.mat
end

end
toc
end

save montecarloresults.mat

%% PROCESS

tic
highVal = zeros(length(variance_pct),scenarios,simlength);
lowVal = zeros(length(variance_pct),scenarios,simlength);

for v = 1:length(variance_pct)
    for i = 1:mcRuns % for each Monte Carlo simulation
        for j = 1:scenarios % for each week-long period
            for k = 1:length(obsTsHours{v}{i}{j}) % for each hour-long period in that week
                if (i==1) || ( tankHead{v}{i}{j}(k) > highVal(v,j,k) )
                    highVal(v,j,k) = tankHead{v}{i}{j}(k); % store highest-so-far calculated tankHead
                end
                if (i==1) || ( tankHead{v}{i}{j}(k) < lowVal(v,j,k) )
                    lowVal(v,j,k) = tankHead{v}{i}{j}(k); % store lowest-so-far calculated tankHead
                end
            end
        end
    end
end

disp(variance_pct(v))

```

```

end

%% CALCULATE RESULTS
calcColor={'m','c','r','g','b'};
for v=1:length(variance_pct)
    for j=1:scenarios
        for i=1:mcRuns
            for k=1:simlength
                % reorganize tankHead so that Monte Carlo runs are indexed last... ugh
                x(v,j,k,i)=tankHead{v}{i}{j}(k);
            end
        end
    end
end
end

tempX = zeros(mcRuns,1);
y = zeros(length(variance_pct),scenarios,simlength,5); % 5 percentiles

for v=1:length(variance_pct)
    for j=1:scenarios
        for k=1:simlength
            for i=1:mcRuns
                % tempX - vector of tankHead values for a single Monte Carlo run
                tempX(i) = tankHead{v}{i}{j}(k);
            end
            % y - matrix of tankHead percentiles organized by (variancepct,scenario,time,:)
            y(v,j,k,:) = prctile(tempX,[0 25 50 75 100]);
        end
    end
end

toc

%% PLOT RESULTS

for v=1:length(variance_pct)
    for j=1:scenarios
        for k=1:simlength
            plotX(k)=obsTsHours{1}{1}{1}(k);
            plotY(k)=(obsTankHead{1}{1}{j}(k)-params(13))*(3.14159*150^2/4*7.48);
            pct0(k)=(y(v,j,k,1)-params(13))*(3.14159*150^2/4*7.48);
            pct25(k)=(y(v,j,k,2)-params(13))*(3.14159*150^2/4*7.48);

```



```

        pct50(k)=(y(v,j,k,3)-params(13))*(3.14159*150^2/4*7.48);
        pct75(k)=(y(v,j,k,4)-params(13))*(3.14159*150^2/4*7.48);
        pct100(k)=(y(v,j,k,5)-params(13))*(3.14159*150^2/4*7.48);
    end

    % plot results in separate figures
    % (shows obs tank levels in the middle & high/low Monte Carlo variations)
    figure(v*scenarios+j)
    hold on
    plot(plotX,plotY,'-','color','k','linewidth',2);
    plot(plotX,pct100,'--','color',[128 128 128]/255);
    plot(plotX,pct75,'-.','color',[64 64 64]/255);
    plot(plotX,pct50,'-','color',[0 0 0]/255);
    plot(plotX,pct25,'-.','color',[64 64 64]/255);
    plot(plotX,pct0,'--','color',[128 128 128]/255);

    xlabel('Time (hr)')
    ylabel('Storage volume (gallons)')
    %title(strcat('Tank level, variance pct=',int2str(variance_pct(v)*100),'%', scenario=',int2str(j)),
    'FontSize',16)
    set(gca,'XLim',[0 168])
    set(gca,'YLim',[0 3e7])
    set(gca,'XTick',[0:12:168])
    set(gca,'YTick',[0:5e6:3e7])
    legend('Observed tank level','Maximum','75th percentile','Mean','25th
percentile','Minimum','Location','Best')
    saveas(gca,strcat('MCv',int2str(variance_pct(v)*100),'s',int2str(j),'.fig'));
    saveas(gca,strcat('MCv',int2str(variance_pct(v)*100),'s',int2str(j),'.png'));
    hold off

    % indicator of whether the observed tank levels are within Monte Carlo variations
    for k=1:simlength
        if obsTankHead{1}{1}{j}(k) >= lowVal(v,j,k) && obsTankHead{1}{1}{j}(k) <= highVal(v,j,k)
            withinLimits(v,j)=1;
        else
            withinLimits(v,j)=0;
            break
        end
    end
end
end
end
save montecarloresultsfinal.mat

```

B.6. enumPumps.m

```
format compact, format short g
close all

lhpQ = [0 1800 3700 5800 7400 8200 9000 10200 11200 ]';
lhpH = [330 310 270 240 220 200 180 140 100 ]';
hsp1Q = [0 4000 6000 6800 8000 8800 11400 ]';
hsp1H = [277 260 235 220 200 190 135 ]';
hsp2Q = [0 2600 3700 4550 5450 6250 6900 7400 8200 ]';
hsp2H = [250 245 240 230 210 195 170 155 115]';
well3Q = [0 2200 2971]';
well3H = [560 500 414]';
well5Q = [0 1000 3000]';
well5H = [505 500 400]';

lhpQe = [0 1800 3700 5800 7400 8200 9000 10200 11200]';
lhpE = [0 32 57 76 84 86 88 83 73]';
hsp1Qe = [0 4000 6000 6800 8000 8800 11400]';
hsp1E = [0 64.3 82 86 89 88 81]';
hsp2Qe = [0 2600 3700 4550 5450 6250 6900 7400 8200]';
hsp2E = [0 62 74 80 84 84 82 78 65]';
well3Qe = [0 1000 2000 2800 3400 3900]';
well3E = [0 54 79 85 80 69]';
well5Qe = [0 1000 2000 2900 3600 4000]';
well5E = [0 50 76 85 80 71]';

%% fit options
fitChar = fitttype('a+b*Q^c',...
    'coefficients',{'a','b','c'},...
    'independent','Q');
fitCharOpt = fitoptions('Method','NonlinearLeastSquares',...
    'Robust','on',...
    'Lower',[0 -1 0],...
    'Upper',[1000 1 5],...
    'StartPoint',[500 -8e-9 2],... 'DiffMinChange',1e-20,...
    'MaxFunEvals',9999,...
    'MaxIter',9999,...
    'TolFun',1e-5,...
    'TolX',1e-15);
```

```

fitEff = fitttype('a*Q^2+b*Q',...
    'coefficients',{'a','b'},...
    'independent','Q');
fitEffOpt = fitoptions('Method','NonlinearLeastSquares',...
    'Robust','on',...
    'Lower',[-9999 -9999],...
    'Upper',[9999 9999],...
    'StartPoint',[1 1],... 'DiffMinChange',1e-20,...
    'MaxFunEvals',9999,...
    'MaxIter',9999,...
    'TolFun',1e-5,...
    'TolX',1e-15);

lhpChar = fit(lhpQ,lhpH,fitChar,fitCharOpt);
lhpEff = fit(lhpQe,lhpE,fitEff,fitEffOpt);
hsp1Char = fit(hsp1Q,hsp1H,fitChar,fitCharOpt);
hsp1Eff = fit(hsp1Qe,hsp1E,fitEff,fitEffOpt);
hsp2Char = fit(hsp2Q,hsp2H,fitChar,fitCharOpt);
hsp2Eff = fit(hsp2Qe,hsp2E,fitEff,fitEffOpt);
well3Char = fit(well3Q,well3H,fitChar,fitCharOpt);
well3Eff = fit(well3Qe,well3E,fitEff,fitEffOpt);
well5Char = fit(well5Q,well5H,fitChar,fitCharOpt);
well5Eff = fit(well5Qe,well5E,fitEff,fitEffOpt);

save fittedcurves.mat lhpChar lhpEff hsp1Char hsp1Eff hsp2Char hsp2Eff ...
    well3Char well3Eff well5Char well5Eff

figure, hold on
plot(lhpChar,lhpQ,lhpH)
plot(lhpEff,lhpQe,lhpE)
hold off

figure, hold on
plot(hsp1Char,hsp1Q,hsp1H)
plot(hsp1Eff,hsp1Qe,hsp1E)
hold off

figure, hold on
plot(hsp2Char,hsp2Q,hsp2H)
plot(hsp2Eff,hsp2Qe,hsp2E)
hold off

```

```

figure, hold on
plot(well13Char,well13Q,well13H)
plot(well13Eff,well13Qe,well13E)
hold off

figure, hold on
plot(well15Char,well15Q,well15H)
plot(well15Eff,well15Qe,well15E)
hold off

%% enumerate pump combos

Qrange = 0:5:40000; % was 0:50:60000
Hrange = 0:1:1500;

lH = lhpChar.a + lhpChar.b * Qrange.^lhpChar.c;
h1H = hsp1Char.a + hsp1Char.b * Qrange.^hsp1Char.c;
h2H = hsp2Char.a + hsp2Char.b * Qrange.^hsp2Char.c;
w3H = well13Char.a + well13Char.b * Qrange.^well13Char.c;
w5H = well15Char.a + well15Char.b * Qrange.^well15Char.c;

lQ = ((Hrange - lhpChar.a)/lhpChar.b).^(1/lhpChar.c);
h1Q = ((Hrange - hsp1Char.a)/hsp1Char.b).^(1/hsp1Char.c);
h2Q = ((Hrange - hsp2Char.a)/hsp2Char.b).^(1/hsp2Char.c);
w3Q = ((Hrange - well13Char.a)/well13Char.b).^(1/well13Char.c);
w5Q = ((Hrange - well15Char.a)/well15Char.b).^(1/well15Char.c);

lE = lhpEff.a*Qrange.^2 + lhpEff.b*Qrange;
h1E = hsp1Eff.a*Qrange.^2 + hsp1Eff.b*Qrange;
h2E = hsp2Eff.a*Qrange.^2 + hsp2Eff.b*Qrange;
w3E = well13Eff.a*Qrange.^2 + well13Eff.b*Qrange;
w5E = well15Eff.a*Qrange.^2 + well15Eff.b*Qrange;

hold on
plot(Qrange,lH,lhpQ,lhpH,'x')

load enumPumps_pumpcurves.mat lH h1H h2H w3H w5H lQ h1Q h2Q w3Q w5Q lE h1E h2E w3E w5E Qrange lhpQ lhpH lhpQe
lhpE

hold on
[ax h1 h2] = plotyy(Qrange,lH,Qrange,lE);

```

```

set(ax(1), 'XLim', [0 1.5e4])
set(ax(2), 'XLim', [0 1.5e4])
set(ax(1), 'YLim', [0 500])
set(ax(2), 'YLim', [0 100])
set(ax(1), 'XTick', 0:2500:15000)
%set(ax(2), 'XTick', 0:2500:15000)
set(ax(1), 'YTick', 0:100:500)
set(ax(2), 'YTick', 0:20:100)
set(ax(1), 'YGrid', 'on')
set(h1, 'linewidth', 2, 'color', 'k')
set(h2, 'line', '--', 'linewidth', 2, 'color', 'k')
pause(1)
set(get(ax(1), 'Ylabel'), 'String', 'Hydraulic head (feet)')
set(get(ax(2), 'Ylabel'), 'String', 'Efficiency (percent)')
set(get(ax(1), 'XLabel'), 'String', 'Flow rate (gallons per minute)')
set(get(ax(1), 'XColor'), 'String', 'k')
[ax2 h3 h4] = plotyy(lhpQ, lhpH, lhpQe, lhpE);
set(h3, 'line', 'none', 'marker', 'x', 'color', 'k', 'markersize', 7);
set(h4, 'line', 'none', 'marker', 'o', 'color', 'k', 'markersize', 7);

%set(ax(2), 'YGrid', 'on')
hold off

%% here goes nothing

vfdStep = 0.2;

w3Qn = zeros(length(Hrange), 1/vfdStep+1, 2);
w5Qn = zeros(length(Hrange), 1/vfdStep+1, 2);
lQn = zeros(length(Hrange), 1/vfdStep+1, 2);
h1Qn = zeros(length(Hrange), 1/vfdStep+1, 2);
h2Qn = zeros(length(Hrange), 1/vfdStep+1, 2);

w3Hn=zeros(length(Qrange), 1/vfdStep+1, 2);
w5Hn=zeros(length(Qrange), 1/vfdStep+1, 2);
lHn=zeros(length(Qrange), 1/vfdStep+1, 2);
h1Hn=zeros(length(Qrange), 1/vfdStep+1, 2);
h2Hn=zeros(length(Qrange), 1/vfdStep+1, 2);

w3En=zeros(length(Qrange), 1/vfdStep+1, 2);
w5En=zeros(length(Qrange), 1/vfdStep+1, 2);
lEn=zeros(length(Qrange), 1/vfdStep+1, 2);

```

```

h1En=zeros(length(Qrange),1/vfdStep+1,2);
h2En=zeros(length(Qrange),1/vfdStep+1,2);

for v=0:(1/vfdStep)
    for k=1:length(Hrange)
        if Hrange(k)<=well3Char.a
            w3Qn(k,v+1,1) = w3Q(k)*v*vfdStep;
        else
            w3Qn(k,v+1,1) = 0;
        end

        if Hrange(k)<=well5Char.a
            w5Qn(k,v+1,1) = w5Q(k)*v*vfdStep;
        else
            w5Qn(k,v+1,1) = 0;
        end

        if Hrange(k)<=lhpChar.a
            lQn(k,v+1,1) = lQ(k)*v*vfdStep;
        else
            lQn(k,v+1,1) = 0;
        end

        if Hrange(k)<=hsp1Char.a
            h1Qn(k,v+1,1) = h1Q(k)*v*vfdStep;
        else
            h1Qn(k,v+1,1) = 0;
        end

        if Hrange(k)<=hsp2Char.a
            h2Qn(k,v+1,1) = h2Q(k)*v*vfdStep;
        else
            h2Qn(k,v+1,1) = 0;
        end
    end

    w3Qn(:,v+1,2) = Hrange*(v*vfdStep)^2;
    w5Qn(:,v+1,2) = Hrange*(v*vfdStep)^2;
    lQn(:,v+1,2) = Hrange*(v*vfdStep)^2;
    h1Qn(:,v+1,2) = Hrange*(v*vfdStep)^2;
    h2Qn(:,v+1,2) = Hrange*(v*vfdStep)^2;
end

```

```

w3Hn(:,v+1,1) = Qrange*v*vfdStep;
w5Hn(:,v+1,1) = Qrange*v*vfdStep;
lHn(:,v+1,1) = Qrange*v*vfdStep;
h1Hn(:,v+1,1) = Qrange*v*vfdStep;
h2Hn(:,v+1,1) = Qrange*v*vfdStep;

for k=1:length(Qrange)
    w3Hn(k,v+1,2) = max([w3H(k)*(v*vfdStep)^2, 0]);
    w5Hn(k,v+1,2) = max([w5H(k)*(v*vfdStep)^2, 0]);
    lHn(k,v+1,2) = max([lH(k)*(v*vfdStep)^2, 0]);
    h1Hn(k,v+1,2) = max([h1H(k)*(v*vfdStep)^2, 0]);
    h2Hn(k,v+1,2) = max([h2H(k)*(v*vfdStep)^2, 0]);

    w3En(k,v+1,1) = Qrange(k)*v*vfdStep;
    w5En(k,v+1,1) = Qrange(k)*v*vfdStep;
    lEn(k,v+1,1) = Qrange(k)*v*vfdStep;
    h1En(k,v+1,1) = Qrange(k)*v*vfdStep;
    h2En(k,v+1,1) = Qrange(k)*v*vfdStep;

    w3En(k,v+1,2) = max([w3E(k), 0]);
    w5En(k,v+1,2) = max([w5E(k), 0]);
    lEn(k,v+1,2) = max([lE(k), 0]);
    h1En(k,v+1,2) = max([h1E(k), 0]);
    h2En(k,v+1,2) = max([h2E(k), 0]);
end
end

%% build combo curves

wCombo = zeros(length(Hrange),6,4);
for i=1:6 % number of wells on
    % build pieces of flow
    a = interp1(w3Qn(:,1/vfdStep+1,2),w3Qn(:,1/vfdStep+1,1),Hrange); % flow in one w3
    b = interp1(w5Qn(:,1/vfdStep+1,2),w5Qn(:,1/vfdStep+1,1),Hrange); % flow in one w5
    % get rid of NaNs that should be 0
    a(isnan(a))=0;
    b(isnan(b))=0;
    % total Q =
    wCombo(:,i,1) = ceil(i/2)*a+floor(i/2)*b;

    % H =

```

```

wCombo(:,i,2) = Hrange;

% build pieces of efficiency
c = interp1(w3En(:,1/vfdStep+1,1),w3En(:,1/vfdStep+1,2),a); % efficiency of one w3
d = interp1(w5En(:,1/vfdStep+1,1),w5En(:,1/vfdStep+1,2),b); % efficiency of one w5
% get rid of NaNs that should be 0
c(isnan(c))=0;
d(isnan(d))=0;
% weighted average E =
wCombo(:,i,3) = (ceil(i/2)*a.*c+floor(i/2)*b.*d)./(ceil(i/2)*a+floor(i/2)*b);

% well parameter for GA =
wCombo(:,i,4) = i;

end

lCombo = zeros(length(Hrange), (3+1)*(1/vfdStep+1)-1,4);
for i=0:3 % non-vfd curves
    for j=0:(1/vfdStep) % vfd curve
        if ~(i==0 && j==0)
            z = i*(1/vfdStep+1)+j;

            if j==0
                % build pieces of flow
                a = interp1(lQn(:,1/vfdStep+1,2),lQn(:,1/vfdStep+1,1),Hrange);
                b = 0;
                % get rid of NaNs that should be 0
                a(isnan(a))=0;
                % total Q =
                lCombo(:,z,1) = i*a;
            else
                % build pieces of flow
                a = interp1(lQn(:,1/vfdStep+1,2),lQn(:,1/vfdStep+1,1),Hrange);
                b = interp1(lQn(:,j+1,2), lQn(:,j+1,1), Hrange);
                % get rid of NaNs that should be 0
                a(isnan(a))=0;
                b(isnan(b))=0;
                % total Q =
                lCombo(:,z,1) = i*a+b;
            end

            % H =

```



```

lCombo(:,z,2) = Hrange;

% build pieces of efficiency
c = interp1(lEn(:,1/vfdStep+1,1), lEn(:,1/vfdStep+1,2), a); % efficiency of one non-vfd
if sum(b) ~= 0;
    d = interp1(lEn(:,j+1,1), lEn(:,j+1,2), b); % efficiency of one vfd
end
% get rid of NaNs that should be 0
c(isnan(c))=0;
d(isnan(d))=0;
% weighted average E =
lCombo(:,z,3) = (i*a.*c+b.*d)./(i*a+b);

% lhp parameter for GA =
lCombo(:,z,4) = i+j*vfdStep;
end
end
end

hCombo = zeros(length(Hrange), (2+1)*(1/vfdStep+1)-1,4);
for i=0:2 % non-vfd curves
    for j=0:(1/vfdStep) % vfd curves (assume both vfds run at same speed)
        if ~(i==0 && j==0)
            z = i*(1/vfdStep+1)+j;

            if j==0
                % build pieces of flow
                a = interp1(h1Qn(:,1/vfdStep+1,2),h1Qn(:,1/vfdStep+1,1),Hrange);
                b = interp1(h2Qn(:,1/vfdStep+1,2),h2Qn(:,1/vfdStep+1,1),Hrange);
                c = 0;
                d = 0;
                % get rid of NaNs that should be 0
                a(isnan(a))=0;
                b(isnan(b))=0;
                % Q =
                if i==1
                    hCombo(:, z, 1) = a;
                elseif i==2
                    hCombo(:, z, 1) = a+b;
                end
            else
                % build pieces of flow

```

```

a = interp1(h1Qn(:,1/vfdStep+1,2),h1Qn(:,1/vfdStep+1,1),Hrange);
b = interp1(h2Qn(:,1/vfdStep+1,2),h2Qn(:,1/vfdStep+1,1),Hrange);
c = interp1(h1Qn(:,j+1,2),h1Qn(:,j+1,1),Hrange);
d = interp1(h2Qn(:,j+1,2),h2Qn(:,j+1,1),Hrange);
% get rid of NaNs that should be 0
a(isnan(a))=0;
b(isnan(b))=0;
c(isnan(c))=0;
d(isnan(d))=0;
% total Q =
if i==0 || i==1
    hCombo(:, z, 1) = i*a+c+d;
elseif i==2
    hCombo(:, z, 1) = a+b+c+d;
end
end
% H =
hCombo(:, z, 2) = Hrange;

% build pieces of efficiency
e = interp1(h1En(:,1/vfdStep+1,1), h1En(:,1/vfdStep+1,2), a); % efficiency of non-vfd
f = interp1(h2En(:,1/vfdStep+1,1), h2En(:,1/vfdStep+1,2), b); % efficiency of non-vfd
if sum(c) ~= 0;
    g = interp1(h1En(:,j+1,1), h1En(:,j+1,2), c); % efficiency of vfd
end
if sum(d) ~= 0;
    h = interp1(h2En(:,j+1,1), h2En(:,j+1,2), d); % efficiency of vfd
end
% get rid of NaNs that should be 0
e(isnan(e))=0;
f(isnan(f))=0;
g(isnan(g))=0;
h(isnan(h))=0;
% weighted average E =
if i==0 || i==1
    hCombo(:, z, 3) = (i*a.*e+c.*g+d.*h)./(i*a+c+d);
elseif i==2
    hCombo(:, z, 3) = (a.*e+b.*f+c.*g+d.*h)./(a+b+c+d);
end

% hsp parameters for GA =
hCombo(:, z, 4) = i;

```

```

        hCombo(:, z, 5) = j*vfdStep;
    end
end

% make sure to get rid of all NaNs
wCombo(isnan(wCombo))=0;
lCombo(isnan(lCombo))=0;
hCombo(isnan(hCombo))=0;

%% plot

close all
hold on
for i=1:size(wCombo,2)
    %plot(wCombo(:,i,1),wCombo(:,i,2))
    ax = plotyy(wCombo(:,i,1),wCombo(:,i,2),wCombo(:,i,1),wCombo(:,i,3));
    set(ax(1), 'XLim', [0 6e4])
    set(ax(2), 'XLim', [0 6e4])
    set(ax(1), 'YLim', [0 700])
    set(ax(2), 'YLim', [0 100])
    pause(1)
end
pause(5)
close all

hold on
for i=1:size(lCombo,2)
    %    plot(lCombo(:,i,1),lCombo(:,i,2))
    %    axis([0 6e4 0 1000])
    ax = plotyy(lCombo(:,i,1),lCombo(:,i,2),lCombo(:,i,1),lCombo(:,i,3));
    set(ax(1), 'XLim', [0 6e4])
    set(ax(2), 'XLim', [0 6e4])
    set(ax(1), 'YLim', [0 700])
    set(ax(2), 'YLim', [0 100])
    pause(1)
end
pause(5)
close all

hold on
for i=1:size(hCombo,2)

```

```

%     plot(hCombo(:,i,1),hCombo(:,i,2))
%     axis([0 6e4 0 1000])
ax = plotyy(hCombo(:,i,1),hCombo(:,i,2),hCombo(:,i,1),hCombo(:,i,3));
set(ax(1), 'XLim', [0 6e4])
set(ax(2), 'XLim', [0 6e4])
set(ax(1), 'YLim', [0 700])
set(ax(2), 'YLim', [0 100])
pause(1)
end
pause(5)
close all

%% now in series

finalCharCurve = zeros(length(Qrange),...
    (size(wCombo,2)-1)*size(lCombo,2)*size(hCombo,2)+(size(lCombo,2)-1)*size(hCombo,2)+size(hCombo,2),...
    7);
for i=1:size(wCombo,2)
    for j=1:size(lCombo,2)
        for k=1:size(hCombo,2)
            z = (i-1)*size(lCombo,2)*size(hCombo,2) + (j-1)*size(hCombo,2) + k;

            % Q =
            finalCharCurve(:,z,1) = Qrange;

            [wComboX, widx] = unique(wCombo(:,i,1));
            [lComboX, lidx] = unique(lCombo(:,j,1));
            [hComboX, hidx] = unique(hCombo(:,k,1));
            wComboY = wCombo(widx,i,2);
            lComboY = lCombo(lidx,j,2);
            hComboY = hCombo(hidx,k,2);

            a = interp1(wComboX,wComboY,Qrange);
            b = interp1(lComboX,lComboY,Qrange);
            c = interp1(hComboX,hComboY,Qrange);
            a(isnan(a))=0;
            b(isnan(b))=0;
            c(isnan(c))=0;

            % H =
            finalCharCurve(:,z,2) = a+b+c;

```

```

        % config cell =
        wComboC = wCombo(widx,i,4);
        lComboC = lCombo(lidx,j,4);
        hComboC1 = hCombo(hidx,k,4);
        hComboC2 = hCombo(hidx,k,5);
        finalCharCurve(:,z,4) = interp1(wComboX,wComboC,Qrange,'nearest');
        finalCharCurve(:,z,5) = interp1(lComboX,lComboC,Qrange,'nearest');
        finalCharCurve(:,z,6) = interp1(hComboX,hComboC1,Qrange,'nearest');
        finalCharCurve(:,z,7) = interp1(hComboX,hComboC2,Qrange,'nearest');
    end
end
end

save enumPumps_pumpcurves.mat

%% Build system curve

% Load calibrated system parameters
load calibration_results3c_manual5.mat bestX3b
pipeParams = bestX2(1:9);

% Build Keq using Hazen-Williams equation - straight from EPANET manual
Keq=0;
for i=1:3
    %
    % H-W C          d in ft          L in ft
    Keq=Keq +4.727 *pipeParams(3*i)^-1.852 *(pipeParams(3*i-2)/12)^-4.871 *pipeParams(3*i-1);
end

% Account for ClearWellOut, GroundTankIn, and GroundTankOut, respectively
Keq=Keq +4.727 *pipeParams(3)^-1.852 *(pipeParams(1)/12)^-4.871 *1;
Keq=Keq +4.727 *pipeParams(6)^-1.852 *(pipeParams(4)/12)^-4.871 *1;
Keq=Keq +4.727 *pipeParams(6)^-1.852 *(pipeParams(4)/12)^-4.871 *1;

% Keq*Q[cfs]=HL[ft] -> Keq*Q[gpm]=HL[ft]. note that H-W flow exponent is 1.852
Keq=Keq*(1/448.831169)^1.852;

% Range of delta z values expected - set to dz from calibrated parameters +
% expected range
dz = 630:1:780;

sysCurve=zeros(length(Qrange),length(dz));
for i=1:length(dz)

```

```

        sysCurve(:,i)=dz(i)+Keq*Qrange.^1.852;
    end

%% Find operating points at intersection of system/char curves

opPts=zeros(length(dz),size(finalCharCurve,2),7); % run time of ~5 minutes
for i=1:length(dz)
    tic
    for j=1:size(finalCharCurve,2)
        % find intersections of Q/syscurve with Q/charcurve
        % Q=          H=
        [opPts(i,j,1) opPts(i,j,2)]=mmcurvex(Qrange',sysCurve(:,i),finalCharCurve(:,j,1),finalCharCurve(:,j,2));

        % calculate weighted efficiency for each operating point
        % wcombo index
        a = floor((j-1)/size(lCombo,2)/size(hCombo,2)) + 1;
        % lcombo index
        b = floor((j-1)/size(hCombo,2)) - size(lCombo,2)*(a-1) + 1;
        % hcombo index
        c = j - size(hCombo,2)*(b-1) - size(lCombo,2)*size(hCombo,2)*(a-1);
        % pare down Q values to unique values
        [wComboX, widx] = unique(wCombo(:,a,1));
        [lComboX, lidx] = unique(lCombo(:,b,1));
        [hComboX, hidx] = unique(hCombo(:,c,1));
        % get efficiency of each pump station
        wComboY = wCombo(widx,a,3);
        lComboY = lCombo(lidx,b,3);
        hComboY = hCombo(hidx,c,3);
        weff= interp1(wComboX,wComboY,opPts(i,j,1));
        leff= interp1(lComboX,lComboY,opPts(i,j,1));
        heff= interp1(hComboX,hComboY,opPts(i,j,1));
        % get head across each pump station
        wComboY = wCombo(widx,a,2);
        lComboY = lCombo(lidx,b,2);
        hComboY = hCombo(hidx,c,2);
        whd= interp1(wComboX,wComboY,opPts(i,j,1));
        lhd= interp1(lComboX,lComboY,opPts(i,j,1));
        hhd= interp1(hComboX,hComboY,opPts(i,j,1));
        % E=
        opPts(i,j,3)= (weff*whd+leff*lhd+heff*hhd)/(whd+lhd+hhd);

        if mod(j,500)==0

```

```

        plot(Qrange',sysCurve(:,i))
        hold on
        plot(finalCharCurve(:,j,1),finalCharCurve(:,j,2),'c')
        plot(wComboX,wComboY,'m')
        plot(lComboX,lComboY,'g')
        plot(hComboX,hComboY,'r')
        scatter(opPts(i,j,1),opPts(i,j,2))
        axis([0 20000 0 1500])
        pause(0.01)
    end

end

hold off
scatter(1,2)

% store pump configuration at each operating point
opPts(i,: ,4)= finalCharCurve(1,: ,4);
opPts(i,: ,5)= finalCharCurve(1,: ,5);
opPts(i,: ,6)= finalCharCurve(1,: ,6);
opPts(i,: ,7)= finalCharCurve(1,: ,7);

toc
end
opPts(isnan(opPts))==0;

%% test plots

hold off
for i=1:100
    scatter(opPts(i,: ,1),opPts(i,: ,3),'x')
    pause(0.1)
end

save enumPumps_almostready20110222_detailed.mat

%% sort by Q ...

load enumPumps_almostready20110222_detailed.mat

clear opPtsFinal
clear opPtsSort

```

```

close all

for i=1:length(dz)
    tic
    % sort by Q and save indexes
    [opPtsSort(i,:,1) sortIdx] = sort(opPts(i,:,1));
    % sort H, E, and the 4 pump parameters
    for z=2:7
        opPtsSort(i,:,z) = opPts(i,sortIdx,z);
    end

    for j=1:size(opPtsSort,2)
        a=1;
        b=1;
        k=1;

        % copy over first operating point
        for z=1:7
            opPtsFinal(i,k,z)=opPtsSort(i,a,z);
        end

        stop=0;
        while stop==0
            b=a;
            c=a;

            % find best operating point out of those which are very close to each other
            flow1=opPtsSort(i,a,1);
            flow2=opPtsSort(i,b,1);
            highestEff=opPtsSort(i,a,3);

            % throw out opPts less efficient than 60% and opPts within 50 gpm of one another
            while (flow2-flow1) < 50 || currentEff < 60
                % oops, reached max flow for this set of opPts
                if b+1 > size(opPtsSort,2)
                    break;
                end

                % iterate flow2
                b=b+1;
                flow2=opPtsSort(i,b,1);
            end
        end
    end
end

```



```

        % compare efficiencies
        currentEff = opPtsSort(i,b,3);
        if currentEff > highestEff
            highestEff = currentEff;
            c=b;
        end
    end

    if opPtsSort(i,c,3)>0
        % copy best operating point to opPtsFinal and iterate k
        for z=1:7
            opPtsFinal(i,k,z)=opPtsSort(i,c,z);
        end
        k=k+1;
    end

    % bring a up to match b
    a=b;

    % check to see if we're done
    if b+1 > size(opPtsSort,2)
        stop=1;
    end
end

end
toc
end

save enumPumps_opPts20110222_detailed.mat

%% plot stuff
close all
for i=1:length(dz)
    hold off
    scatter(opPtsFinal(i,:,1),opPtsFinal(i,:,2),'o')
    hold on
    scatter(opPtsSort(i,:,1),opPtsSort(i,:,2),'x')
    scatter(opPtsFinal(i,:,1),opPtsFinal(i,:,3),'o')
    scatter(opPtsSort(i,:,1),opPtsSort(i,:,3),'x')
    axis([0 20000 0 1000])
    pause(0.1)
end

```

B.7. optimizeGA.m

```
format compact; format short g;

%% INITIAL SET UP

global shouldGraph
shouldGraph = 1;
global suppressWarning
global suppressAnnoyingWarning
suppressWarning=1; suppressAnnoyingWarning=1;
warning off MATLAB:loadlibrary:typenotfound

global opPtsFinal
if isempty(opPtsFinal)
    load enumPumps_opPts20110222_detailed.mat opPtsFinal
end

global bestX3b
if isempty(bestX3b)
    load calibration_results3c_manual5.mat bestX3b
end

global lhpEff hsp1Eff hsp2Eff well3Eff well5Eff %#ok<NUSED>
load fittedcurves.mat
%efficiencyCurves = { lhpEff hsp1Eff hsp2Eff well3Eff well5Eff };

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

timeStep = 6; % hours - this is the optimization time step not the hydraulic time step
endTime = 24; % hours
optSwitchTimes = 0:timeStep:endTime;
forceDuration=endTime/24;
tankDiameter=150; % for converting between storage volume and head

%% PREPARE ORIGINAL SCENARIOS FOR OPTIMIZATION

inpt(1:13)=bestX3b(1:13);

obsFile={'2009peak2.csv','2009peak.csv','2009rise.csv','2008fall.csv','2007flat.csv'};
```

```

for scen=1:5
    load(strcat(obsFile{scen},'.mat'))
    obsPatAll(scen, :, :) = obsPat;
    obsPatAll(scen, :, 2) = obsPatAll(scen, :, 2) * baseDem;

    i=1;
    % grab observed demands and tank levels at each time step
    while obsPatAll(scen, timeStep*(i-1)+1, 1) - 1 <= (endTime/24)
        lowres_obsPat(scen, i, 1) = obsPatAll(scen, timeStep*(i-1)+1, 1); %#ok<SAGROW>
        lowres_obsPat(scen, i, 2) = mean(obsPatAll(scen, timeStep*(i-1)+1:i*timeStep, 2)); %#ok<SAGROW>
        lowres_obsPat(scen, i, 3) = obsPatAll(scen, timeStep*(i-1)+1, 3); %#ok<SAGROW>
        i=i+1;
    end

    %for i=1:5
        inpt(14+4*(scen-1)) = round(bestX3b(14+4*(scen-1))); % no-vfd well pumps
        inpt(15+4*(scen-1)) = bestX3b(15+4*(scen-1));
        inpt(16+4*(scen-1)) = round(bestX3b(16+4*(scen-1))); % no-vfd HSPs
        inpt(17+4*(scen-1)) = bestX3b(17+4*(scen-1));
    %end
    % load the right pump parameters
    pumpParam = [inpt(14+4*(scen-1)) inpt(14+4*(scen-1)+1) ...
        inpt(14+4*(scen-1)+2) inpt(14+4*(scen-1)+3)];

    [origTankError(scen), origP(scen), temp2, temp3, origTankHead(scen, :), origPumpStats(scen, :, :, :)] ...
        = cityWdsRun(scen, bestX3b(1:13), pumpParam, [], [], [], forceDuration); %#ok<SAGROW>

    totalEnergyUsed(scen) = sum(sum(origPumpStats(scen, :, :, 6))); %#ok<SAGROW>
    perpumpEnergyUsed(scen, :) = sum(origPumpStats(scen, :, :, 6)); %#ok<SAGROW>
end

save optimization_resultspt1.mat
close all

%% BEGIN GA LOOP

for optScen=1:5

    %% PREPARE FORECASTS AND PENALTY MEASURES

    % read in the actual demands & tank levels from the observed scenarios
    optForecast{optScen} = [obsPatAll(optScen, :, 1)' obsPatAll(optScen, :, 2)'];

```

```

lowres_optForecast{optScen} = [lowres_obsPat(optScen,:,1)' lowres_obsPat(optScen,:,2)'];

% need to convert the observed tank volumes to levels above minimum tank elev
obsTankLvls(optScen,:) = obsPatAll(optScen,:,3)/(3.1416*tankDiameter^2/4*7.48);
obsTankLvls(optScen,:) = obsTankLvls(optScen,:)+bestX3b(13);

% [beginning, ending, minimum] tank levels
% (ending and minimum are used for assessing penalties)
optTankLvls(optScen,:) = [ ...
    obsTankLvls(optScen,1)... % initial level
    obsTankLvls(optScen,length(obsTankLvls(optScen,:)))... % ending level
    min(obsTankLvls(optScen,:))]; % minimum observed

%% PREPARE GA

useguess = 0; % otherwise it'll pick a random population

% increase forecasted demand by 5% to give it room to bring pumped flow
% down without too many penalties
guess = lowres_optForecast{optScen}(:,2)*1.05;

lb = zeros(1,endTime/timeStep);
ub = 21958*ones(1,endTime/timeStep); % 21958 gpm is the max flow from opPts

dvs = length(optSwitchTimes)-1;

% GA parameters
popSize = 100;
gens = 30;
trials = 2;
crossRate = 0.6;
mutRate = 0.3;

%% EXECUTE GA

tic

for j = 1:trials

    %close all
    options = gaoptimset('PopulationSize',popSize);
    options = gaoptimset(options,'PopInitRange',[lb;ub]);

```

```

options = gaoptimset(options, 'Generations',gens);
options = gaoptimset(options, 'CreationFcn',@gacreationuniform);
options = gaoptimset(options, 'CrossoverFraction',crossRate);
options = gaoptimset(options, 'CrossoverFcn',@crossoverarithmetic);
options = gaoptimset(options, 'MutationFcn',{@mutationuniform,mutRate});
%options = gaoptimset(options, 'TolFun',1e-1);
options = gaoptimset(options, 'StallGenLimit',10);
options = gaoptimset(options, 'Display','iter');
options = gaoptimset(options, 'PlotFcns', @gaplotbestf);

if useguess==1
    options = gaoptimset(options,'InitialPopulation', repmat(guess(1:dvs)',popSize,1));
end

[x(optScen,j,:) y(optScen,j) exitflag(optScen,j) output(optScen,j)] = ...
    ga(@(inpt)cityWdsOpt(optForecast{optScen},...
        [optSwitchTimes; [inpt 0]]',optTankLvls(optScen,:)),...
        dvs,','',',',',',lb,ub','',options); %#ok<SAGROW>

toc

save optimization_results2.mat

% save GA plots
figs=get(0,'children');
for f=1:length(figs)
    saveas(figs(f),['optimization2pt1_fig' num2str(f) '_scen' num2str(optScen) '.fig'])
    close(figs(f))
end

end

% find the best GA result out of the trials
[bestTrial, bestIndex] = min(y(optScen,:));
bestX(optScen,:) = x(optScen,bestIndex,:); %#ok<NOPTS>

% calculate energy used in the best GA result
shouldGraph = 0;
%[a b c d e ]=...
[lowestY(optScen) p(optScen) tankHead{optScen} pumpStats{optScen} dem{optScen} ] = ...
    cityWdsOpt(optForecast{optScen},...
    [optSwitchTimes; [bestX(optScen,:) 0]]',optTankLvls(optScen,:));

```

```
lowestY
shouldGraph = 1;
save optimization_results2.mat
end
```

B.8. cityWdsOpt.m

```
function [totalEnergy, p, tankHead, pumpStats, dem] = ...
    cityWdsOpt(optForecast,optPumpFlow,optTankLvls)

% When calculating energy of optimized pumped flows, need three input variables
% optForecast (2 by _) - forecasted demand, t in hrs vs Q in gpm
% optPumpFlow (2 by _) - pumped flows, t in hrs vs Q in gpm
% optTankLvls (1 by 2) - start tank level, target end level, minimum level

% Initialize variables
global shouldGraph %#ok<NUSED>
global bestX3b

    params = bestX3b(1:13);

    [tankError, p, dummy, dummy2, tankHead, pumpStats, dem] = ...
        cityWdsRun(1,params,[],optForecast,optPumpFlow,optTankLvls);

% Calculate total energy used
totalEnergy = sum(sum(pumpStats(:, :, 4)));
totalEnergy = totalEnergy+p;
```

VITA

John Garrett Johnston was raised and home-schooled in Victoria, Texas. In 2004, he moved to College Station, Texas to attend Texas A&M University. Five years and three internships later, Garrett graduated with a Bachelor of Science in Civil Engineering in 2009. Since then, he has pursued a Master of Science in Civil Engineering at Texas A&M University, with a focus on water resources engineering.

Garrett's graduate studies at Texas A&M have been principally supported by the Zachry Department of Civil Engineering Haynes '46 Fellowship and a teaching assistantship under Dr. Kelly Brumbelow. During his undergraduate studies, Garrett was fortunate to have the opportunities to intern with Goodwin & Marshall, Inc. in Grapevine, Texas; Landtech Consultants, Inc. in Victoria, Texas; and Bleyl & Associates in Bryan, Texas. The experiences and relationships formed at these companies were instrumental in developing his interest in water resources engineering prior to his enrollment in graduate school. Garrett is now pleased to be working for Freese & Nichols, Inc. as a civil engineer in the stormwater management division.

Garrett can be reached by e-mail at garrett@tamu.edu. He can also be reached through Dr. Kelly Brumbelow at the Zachry Department of Civil Engineering, Wisenbaker Engineering Research Center Room 205-L, 3136 TAMU, College Station, Texas 77843-3136.